Universitat de Lleida

TREBALL FINAL DE GRAU

Estudiant: **Miquel Rosell Tarragó**

Titulació: Grau en Enginyeria Mecànica

Títol de Treball Final de Grau: Design of a 3D photogrammetry acquisition system and data processing workflow automation

Director/a: Àlex Escolà i Ricardo Sanz

Presentació

Mes: Juliol

Any: 2020

ACKNOWLEDGEMENTS

I would like to thank my family and friends for all the help and support given while making this project.

I would also like to thank my supervisors Dr. Àlex Escolà and Dr. Ricardo Sanz for their invaluable guidance, help and encouragement, as well as Dr. Eduard Gregorio, Dr. Jordi Gené, Dr. Jaume Arnó and Dr. Joan Ramón Rosell.

For all the provided help with Meshroom, the used software, I extend special thanks to GitHub users “natowi” and “faviencastan”.

Finally, I would like to thank God Almighty for giving me the knowledge, ability, strength and opportunity to undertake this research study and to persevere and complete it satisfactorily. Without his blessings, this achievement would not have been possible.

This work was partly funded by the Spanish Ministry of Science, Innovation and Universities (project PAgFRUIT: Precision agriculture technologies to optimize canopy management and sustainable crop protection in fruit orchards. Project code: RTI2018-094222-B-I00).

ABSTRACT

The objective of this final degree project was to develop a mechanical-optic system for acquiring photographic images, alongside a posterior automatic processing step based on photogrammetric techniques to generate three-dimensional models. The methodology had to be optimized for industrial and agricultural applications.

To achieve this objective, an investigation of the state-of-the-art of the photogrammetry field was done in order to find, discuss and optimize the available methods. Further investigation lead to the specification of the project variables regarding the mechanical-optic system.

Due to the consequences of Covid-19, the planned experimental field tests were substituted by simulations using 3-D software. The change turned out to be a good addition to the project, since it allowed a controlled variable study and discussion.

At the end of the project, an optimal camera array configuration was found in order to obtain an accurate and fast scanning rig. The camera array has all the cameras in a vertical concave array around the scanning subject. It was also found that filtering the dense point clouds increased the computing time by several hours, but the error in the results was not reduced significantly, and some information was lost in the process.

A specific node tree workflow for the used software was also found. This configuration allowed the users to have a fully automated processing step, while still letting room for customization based on the requirements of the project.

TABLE OF CONTENTS

TABLE OF CONTENTS: FIGURES

TABLE OF CONTENTS: TABLES

INTRODUCTION

At present, lots of procedures involving 3-D data or models are moving from traditional methodologies to digital workflows, due to the many advantages they offer, ranging from easier prototyping and planning to better and more accurate solutions.

The mentioned workflows are great for creating projects from scratch, but sometimes there needs to be a reference or starting point from the real world, be it an object, a building, etc. In these cases, we need a method to transfer the object to the digital system so it can be processed alongside all the other components, like what sensors do to monitor variables.

One of the ways of doing so is by manually creating the 3-D model of the existing object from scratch by using different kinds of CAD or modelling software. This can be easily done for simple objects if one has the correct measuring tools and knowledge. The problem comes when the object has complex shapes, like organic objects or statues, or when the dimensions are out of our measuring range. In those cases, 3-D scanners can be used to get the desired results.

Their operation principle could be compared to text scanners: when we need to digitize a small text, it can sometimes be quicker to manually type it. But when we have a long sequence of characters the best solution is usually to use scanning technology.

One of the problems of these scanners is their price, since they usually represent a large investment for the user due to the components needed to manufacture them, the calibration, the required software and other factors.

There is an alternative to conventional 3-D scanners called “Photogrammetry”, which refers to the science of making reliable measurements using photographs. This is accomplished by using different algorithms to detect what is called “features” in the images. These identified features can then be cross-referenced between pictures taken from different angles so they can be used as reference points to stitch the images together, obtaining a bigger image or mosaic (used, for example, in the creation of panoramic photos). Photogrammetry software make use of the detected features to estimate the shapes of scanned objects in the 3-D space.

With the mentioned method we can obtain a 3-D model based on photos taken by any device, if we have some additional information about its optical specifications. So, the price of the system required to capture the information is cut down drastically.

But using photogrammetry also introduces some setbacks. Mainly, the need of a good software to process the pictures and the large number of files needed for any project, from lots of input images to all the files generated by each step of the data processing chain. The user also needs to manually take the photographs, then move them from the camera to the computer, then import them into their software of choice and then process them to obtain the results. This process can be complicated or time consuming or delicate when working on large or numerous projects.

Photogrammetry is usually used in several fields: (AliceVision, 2020)

- Cartography, including extra-terrestrial
- Tourism
- Civil Engineering - Urban Planning
- Architecture
- Cultural heritage - Archaeology (like CyArk)
- Robotics
- Military - Aerial Surveillance
- Medical (like shape-from-template)
- Disasters (fast mapping)
- Manufacturing Industry
- Post-Production - Visual Effects (VFX)

With these aspects in mind, the main goal of this final degree project is designing a system to completely automate the scanning workflow from the image capturing to the final 3-D model or, in this case, 3-D point cloud.

To achieve this, I will design an armature system to support a camera rig, allowing the mounting of the system to different displacement solutions, from a motorized vehicle to a manual kart. I will also study the optical parameters and choose the most appropriate capturing devices so they can work with an optimal image overlapping without the need for manual adjusting.

I will also automate the image capturing process, so the user only needs to change the camera rig position while the rest of the workflow is fully automated, sending the images automatically to the processing unit, importing them inside the program and taking care of all the output files.

The system will be tested in the agronomic and industrial fields, scanning tree plantations and tube installations respectively.

1. THEORETICAL FRAMEWORK

1.1.3-D scan

A 3-D scan is a digital representation of an object, that can include or not its color data. This representation can usually have four information levels in the field of photogrammetry:

- **Sparse Point Cloud:** A digital representation of the position in a 3-D space of the points that define the main features of the object. These can represent edges, holes or cavities, etc. The main advantage of this kind of representation is the lower processing time required to obtain good results, compared to the other information levels. It is useful for rough estimations or to check if the scan has some blind spots that need to be rescanned. A sparse point cloud can also store color information for each digital point, based on the real color taken from the images.

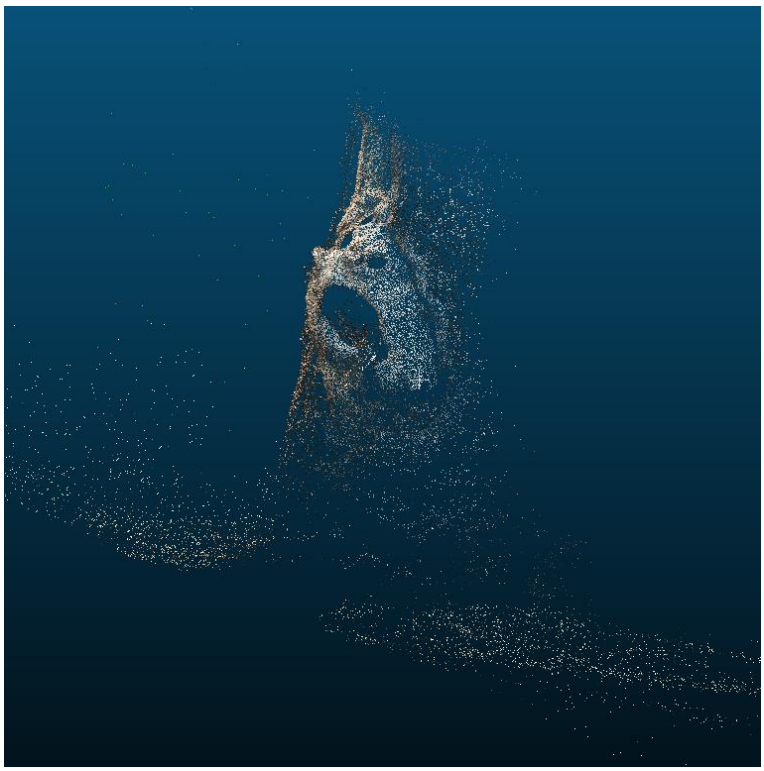


Figure 1 - Sparse point cloud inside Cloud Compare

- **Dense Point Cloud:** A digital representation of the position of all the points that define the scanned scene, usually after some filtering made by the software to reduce noise in the output file. In this case, for example, we can see the entirety of points that make up a flat surface, while in a sparse representation we usually only see the edges of the mentioned surface. This solution offers a lot of advantages, since it is a representation of the scanned scene with all the information that the used software was able to obtain with the supplied images. For comparison, you can get about 5 million points in a dense point cloud based on the same scene as a sparse point cloud that will output about 7000 points. The main disadvantage of this kind of representation is the extremely high processing time needed, especially if the user does not have powerful equipment. It is worth mentioning that the extra points obtained in a dense point cloud are not an interpolation between the ones available in a sparse representation. They are points that add new information about the real scene to the output.
  As it happens with sparse representations, these points can also store the colors associated to their corresponding position in the scanned object.

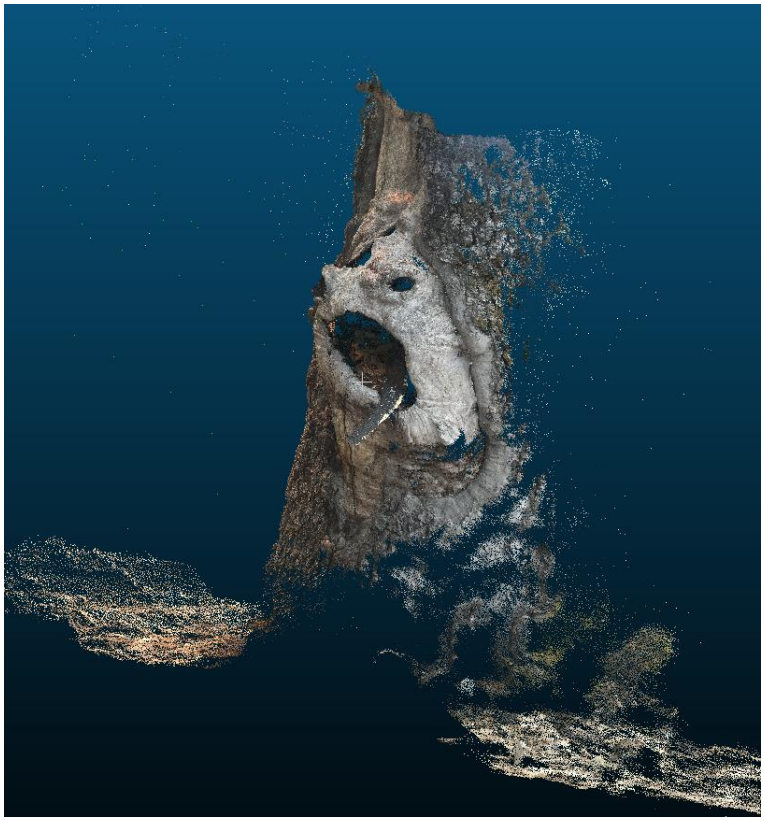


Figure 2 - Dense point cloud inside Cloud Compare

- **Mesh:** A digital representation of the scanned object with a defined surface, usually based on the information provided by the dense point cloud. The term “solid” is the key characteristic of this type of information level, since the previously mentioned outputs lack a defined volume. Point clouds, as the name implies, are a group of individual and disconnected points that have a specific location assigned. In a 3-D mesh, the points are connected creating edges, usually following a triangulated structure. This gives a volume (in the case of a fully closed surface) and a more defined shape to the scan. This type of

representation usually doesn't have any color information, since each software can assign a specific color or material to the mesh. The main disadvantage of this kind of representation is that, depending on the software and the method used for the mesh reconstruction, the mesh can lose fidelity and/or accuracy in the shape and dimensions of the scanned object, since the software must figure out the connections between the digital points. This can lead to unwanted errors and artefacts when the points are close together or when the scan has complex shapes. The time needed for this kind of representation is similar to the dense point clouds, but it takes a little bit longer since the software needs to first create the point cloud and then assign edges to it using another algorithm.

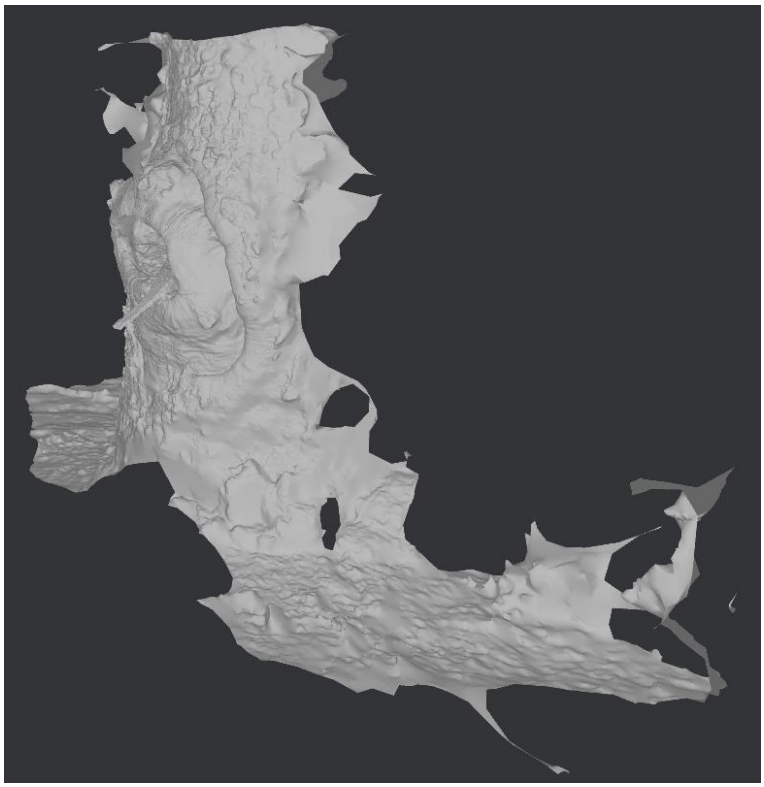


Figure 3 - Mesh inside Meshroom

- **Textured mesh:** A mesh representation that has the color information from the real scanned object. With a textured mesh, the color information captured is superior to the digital representations mentioned before. In the previous methods, each point/vertex is assigned a color, so the color information is limited to the number of points in the point count. With this method, an actual image of the object is wrapped around the 3-D mesh/surface, adding all the detail available even if the point count is not high enough. This way the user can obtain, for example information on the texture of a tree bark without needing a really detailed scanned geometry.
  The process is like the one used to create a paper cube but, instead of creating a 3-D object from a flat surface, the 3-D object is unwrapped/unfolded onto a flat plane. In order to do this, there needs to be one or multiple "cuts" in the geometry for it to be able to unfold. The method that provides the best results is to manually define the cuts,

but 3-D software usually have an option to detect the zones that may need them automatically.

Once the 3-D geometry is unwrapped, the software deforms the image from the real object, so it fits the obtained perimeter. This image is created using different algorithms to stitch the images taken from several angles into one that covers all the views, like a panorama shot.

The time needed for this kind of representation is similar to the uncolored mesh, but it takes a little bit longer since the software needs to first create it and then unwrap and assign the image.

Figure 4 - Textured mesh inside Meshroom

The last three mentioned information levels have a sequential hierarchy. This means that you need to have the previous digital representation processed in order to obtain a specific representation, as seen in Figure 5: Sequential hierarchy of 3-D model representation (To obtain the mesh you need to have the Dense Point Cloud processed previously, for example):

Figure 5: Sequential hierarchy of 3-D model representation

1.2. Photogrammetry study

1.2.1. Camera parameters

Some ISO and shutter speed parameters must be set as a starting point. Then, if the scene is too dark, ISO can be increased, or the shutter speed can be decreased too. A tripod will be needed in this case due to the slow shutter speed.

The usage of a full-frame camera will allow higher ISOs.

Once the starting parameters are set, fine tuning is needed based on the actual scene. It is better to start from an angle where a lot of detail can be seen, since the photographs are usually processed in order. (Blizard, 2020)

1.2.2. Image Feature Extraction algorithms

All the information in this section is extracted/based from (Saleem, 2018).

Image Feature Extraction algorithms are a crucial part of photogrammetry, since the detected image features are then matched and used to process the sparse point cloud of the scan, using the Structure from Motion method, as it will be explained later.

First, a brief explanation of the different algorithms is needed to have an idea of how they work:

- **SIFT**

The most renowned feature-detection-description algorithm is the Scale Invariant Feature Transform (SIFT), which has a high computational cost but ensures a very high adaptability to image scale, rotations, and limited affine variations.

- **SURF**

Speeded Up Robust Features (SURF) is an algorithm that has a low computational cost compared to SIFT, while remaining invariant to rotation and scale changes in the images. The detected features have little affine invariance

- **KAZE**

KAZE introduce a moderate increase in computational time, and in addition to rotation invariance they are also invariant to limited affine, rotation and scale.

- **AKAZE**

This algorithm is based on the same principle as KAZE, but it uses a more computationally efficient framework. The detected features are invariant to limited affine, rotation and scale, and at varying scales they have more distinctiveness.

- **ORB**

The features detected by the Oriented FAST and Rotated BRIEF (ORB) algorithm are invariant to limited affine changes, scale and rotation.

- **BRISK**

BRISK features are invariant to limited affine changes, rotation and scale.

1.2.3. Algorithm comparison

Based on their ability to detect high quantity of features, the generic order of feature-detector-descriptors is:

ORB > BRISK > SURF > SIFT > AKAZE > KAZE

Based on their computational efficiency per feature-point, the generic order of feature-detector-descriptors is:

ORB > ORB (1000) > BRISK > BRISK (1000) > SURF (64D) > SURF (128D) > AKAZE > SIFT > KAZE

Based on their feature-matching efficiency per feature-point, the generic order of feature-detector-descriptors is:

ORB (1000) > BRISK (1000) > AKAZE > KAZE > SURF (64D) > ORB > BRISK > SIFT > SURF (128D)

Based on their total image matching speed, the generic order of feature-detector-descriptors is:

ORB (1000) > BRISK (1000) > AKAZE > KAZE > SURF (64D) > SIFT > ORB > BRISK > SURF (128D)

For this specific project the interest lays in obtaining good results over fast processing times, so this will determine the range of algorithms used. Regarding the different applications, for agricultural scans the camera rig will be moving in a straight line with constant angle, so rotation adaptiveness is not important. The industrial application will need a better handling of different camera angles, so the algorithm may need to be changed.

In conclusion, SIFT feature-detector-descriptor will be used inside the photogrammetry software, since ORB and BRISK are not available and SIFT presents the best overall accuracy.

An extensive algorithm comparison is available in Annex 1 and (Saleem, 2018).

1.2.4. Structure from motion (SfM)

Structure from motion is a photogrammetric technique used to estimate three-dimensional structures based on several 2-D images. The base principle for this technique is called motion parallax, where the objects around an observer move different amounts as he moves, depending on their distance to the observer. Humans use this to perceive a lot of 3-D information from their surroundings. (Wikipedia, 2020)

To obtain a good 3-D representation, the correspondence between the reconstruction of the 3D object and the used images needs to be found. These correspondences are found using the different algorithms explained in Section 2.2.2 to detect important features in the images, such as corners. These features are then tracked from one image to the next, and the images are matched. Sometimes some of the matched features are incorrectly matched, so a filter must be used.

The trajectories followed by the features over time are used to reconstruct their location in a 3-D space, alongside the camera's position and motion.

1.2.5. Photogrammetry capturing basics (Meshroom, 2020)

Photogrammetry scanning is a very interesting method because it usually is way cheaper than other 3-D scanning methods, with results that have a comparable quality. In order to obtain these, it's important to use a proper technique to maximize the scan fidelity:

- The scene where the object is located should be well lit.
- Avoid hard shadows (since they don't have any data for feature detection), reflections (since they produce error during the reconstruction because the reflection changes as the camera moves) and transparent objects.
- It's better to take the pictures in indirect light, such as in the daylight shadow of a building or during a cloudy day.
- It's important to void plain and one-colored surfaces, since they have too much similar points and the detected features can get mixed.
- Do not use a flash.
- Usually it's recommended to never change the focal length while shooting, or to use a fixed lens. Since Meshroom, the software used, can work with images of different characteristics and from different cameras, this is not an important issue. Taking several images with each focal length is still recommended.
- When possible, obtain images from enough angles so that the entire object can be seen (change the camera perspective).
- It's critical to avoid any moving objects in the scene or in background.
- When using the turnaround method (rotating the subject instead of rotating the camera around it), it's important that the background is as plain as possible. A white sheet can be used, for example.
- The object of interest should always fill most of the image in order to obtain good results.
- The side overlap must be min. 60%, and the frontal overlap min. 80%.
- For each shot, move to a new position (or rotate the object, depending on the used method). Do not take multiple images from the same spot, as they don't add any new information.
- Taking photographs multiple times following different patterns to leave no blind spots is always a good idea.
- Avoid shaking or blurred images.
- The more images in the database, the better. It's better to decide not to use them later than having to return to the place where the subject is in order to fill some blind spots. In this case, Meshroom can determine which images add enough new information, and which ones can be discarded automatically.

2. MATERIALS AND METHODS

2.1. Variable analysis

2.1.1. Device discussion

For this project, three Android smartphones will be used as scanning devices, since their processing power is higher when comparing to DSLR or more traditional cameras and the sensors can have higher pixel outputs per shot. Additionally, they can be a lot cheaper than compared to standard cameras, while offering better specifications for photogrammetry. To get bokeh/blur free pictures on a DSLR it is almost mandatory to buy an additional (usually expensive) lens, while smartphone lenses don't present any background blur in normal shooting conditions. This increases the cost of using DSLR cameras even more.

As an overall note, standard, or DSRL, cameras can output more beautiful pictures. But, for photogrammetry, the interest is not put into beautiful photos, but into photos with clarity, dynamic range and high pixel counts.

To ensure a focused shooting session, we need to take a test shot and zoom in to 100% on the camera's display. To make sure that every part of the subject is in focus, the nearest part of the subject can be compared to the farthest one. Then we can proceed taking some more test photos, to ensure that the subject is in focus, and that the ISO is not making the image too noisy. (Blizard, 2020)

2.1.2. Displacement velocity and inter-photo distance

As discussed in Section 1.2.5, a minimum of β=0.8 overlapping between the pictures taken is needed in order to get a good photogrammetry result. So, the maximum distance between each shot (d_{max}) needs to be calculated. This distance will determine the maximum speed (v_{max}) at which the aluminum armature with the camera rig can be moved. To calculate it, the distance between the cameras and the object (h), and the angle of view (α_{vision}) need to be known, where the angle of view corresponds to:

$$\alpha_{vision} = 2arctan\frac{d}{2f} \quad (1)$$

Where "d" represents the size of the film (or sensor) in the direction measured. For example, for 35 mm film (which is 36 mm wide and 24 mm high), d=36 mm would be used to obtain the horizontal angle of view and d=24 mm for the vertical angle.

"f" represents the effective focal length.

With all the known variables, the distance can be calcualted using Formula (2):

$$d_{max} = 2 \cdot h \cdot tan\left(\frac{\alpha_{vision}}{2}\right) - overlap = 2 \cdot h \cdot tan\left(\frac{\alpha_{vision}}{2}\right)[1 - \beta] \quad (2)$$

With the maximum distance between shots determined, the maximum speed can be calculated by using Formula (3):

$$v_{max} = \frac{d_{max}}{t_{shots}} = \frac{2 \cdot h \cdot tan\left(\frac{\alpha_{vision}}{2}\right)[1-\beta]}{t_{shots}} \quad (3)$$

In the specific case of this project, the known camera parameters will be:

	Sensor Size	Focal Length (Equivalent)	Aperture/F-stop
Main Lens	5.76 x 4.29 mm	26 mm	1.75
Tele Lens	4.23 x 3.17 mm	50 mm	2.2
Wide Lens	4.66 x 3.5 mm	16 mm	2.2

Table 1 - Camera parameters

The view angles will be:

	View Angle (Horizontal)	View Angle (Vertical)
Main Lens	12.64	9.43
Tele Lens	4.84	3.63
Wide Lens	16.57	12.48

Table 2 - View angles

The maximum distance between shots will be:

	Main Lens	Tele Lens	Wide Lens
d_{max} (Horizontal)	0.13 m	0.051 m	0.175 m
d_{max} (Vertical)	0.1 m	0.035 m	0.131 m

Table 3 - Maximum distance between shots

The maximum speed in this specific case will be, then:

	Main Lens	Tele Lens	Wide Lens
v_{max} (Horizontal)	0.13 m/s	0.051 m/s	0.175 m/s
v_{max} (Vertical)	0.1 m/s	0.035 m/s	0.131 m/s

Table 4 - Maximum camera speed

Considering that, the devices can take one image per second when using HDR mode.

2.2.Software: Meshroom

2.2.1. Software introduction

“Meshroom is a free, open-source 3-D Reconstruction Software based on the AliceVision framework.

AliceVision is a Photogrammetric Computer Vision Framework which provides 3-D Reconstruction and Camera Tracking algorithms. AliceVision comes up with strong software basis and state-of-the-art computer vision algorithms that can be tested, analyzed and reused. The project is a result of collaboration between academia and industry to provide cutting-edge algorithms with the robustness and the quality required for production usage.” (Meshroom, 2020).

2.2.2. Node tree explanation

For this project, the node tree shown in Figure 7 was used. The tree starts with the CameraInit node, that loads the image metadata and sensor information into the data flow. This data then gets sent to four nodes: FeatureExtraction, ImageMatching, FeatureMatching and StructureFromMotion.

FeatureExtraction node uses the algorithms explained in Section 2.2.2 to extract distinctive groups of pixels that are, to some extent, invariant to changing camera viewpoints during image acquisition in the imported images, using the CameraInit node outputs. A feature detected in the scene should have similar feature descriptions in all imported images (AliceVision, 2020). These extracted features are then sent to three nodes: ImageMatching, FeatureMatching and StructureFromMotion.

ImageMatching node uses the outputs from CameraInit and FeatureExtraction to determine mutual parts of the imported images, taken from different positions. These matched images are sent to the FeatureMatching node. (AliceVision, 2020)

This node then takes its inputs from CameraInit, FeatureExtraction and ImageMatching, as shown in Figure 7, and uses them to find corresponding features between different candidate image pairs (ArcGIS Pro, 2020) (AliceVision, 2020). The output is sent to the StructureFromMotion node.

“The objective of this next step is to understand the geometric relationship behind all the observations provided by the input images and infer the rigid scene structure (3-D points) with the pose (position and orientation) and internal calibration of all cameras.” (AliceVision, 2020)

The StructureFromMotion node then generates the Sparse Point Cloud using the obtained data from the previous steps, and sends it to 5 nodes: PrepareDenseScene, DepthMap, DepthMapFilter, Meshing and Texturing.

The PrepareDenseScene node undistorts the imported images and generates EXR image files (Meshroom, 2020), that will be used for DepthMap and Texturing.

The DepthMap node then generates the depth map for the 3-D object reconstructed from the imported image dataset (AliceVision, 2020). These maps are an image or image channel that contains information relating to the distance of the surfaces of scene objects from a viewpoint (Wikipedia, 2020). In this step there's a parameter called "Downscale" that can be used to control the resolution of the depth map images. This resolution has a direct impact on the number of points in the dense point cloud in the output. By default, it is set to 2, which means it divides the resolution of the input images by 2 in order to create the depth maps. By setting it to 1, it divides the resolution by 1, so Meshroom works with all the available information provided by the input images. This way, the number of output points could be doubled. This solution was provided by GitHub user "natowi", and it can be seen in Figure 6:

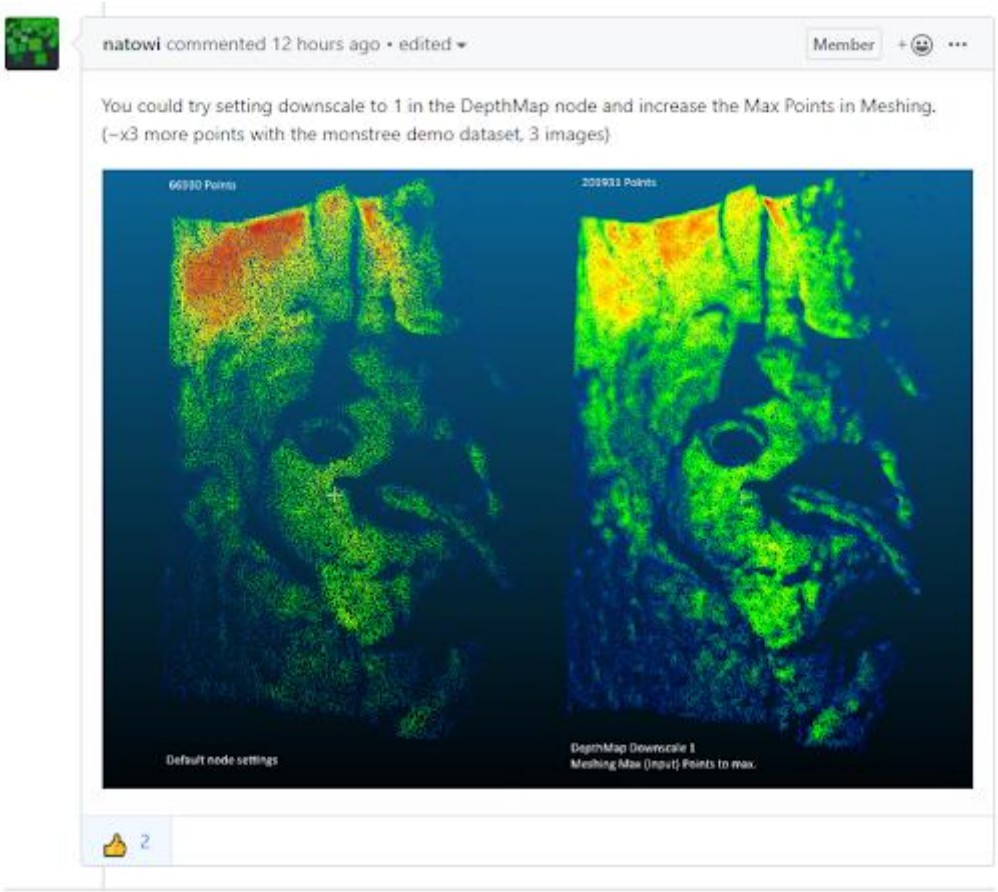


Figure 6. Proposed solution to increase the number of output points, and a comparison between Downscale = 2 and Downscale = 1

During the processing, some generated depth maps will have visible areas that may be occluded by other depth maps. The DepthMapFilter node isolates these areas, using the StructureFromMotion and DepthMap data, to force depth map consistency, since the original depth maps will not be entirely consistent. (Meshroom, 2020)

The Meshing node uses the StructureFromMotion, DepthMap and DepthMapFilter nodes as inputs to generate a dense geometric surface representation of the subject. The mentioned node uses the provided data to generate a Dense Point Cloud first, and then makes an interpretation of its geometry to join all the points from the cloud with the ones near them using edges. It is in this node, then, that we can obtain the Dense Point Cloud file needed for the project. This node also has the option to generate a point cloud with color information embedded in each point.

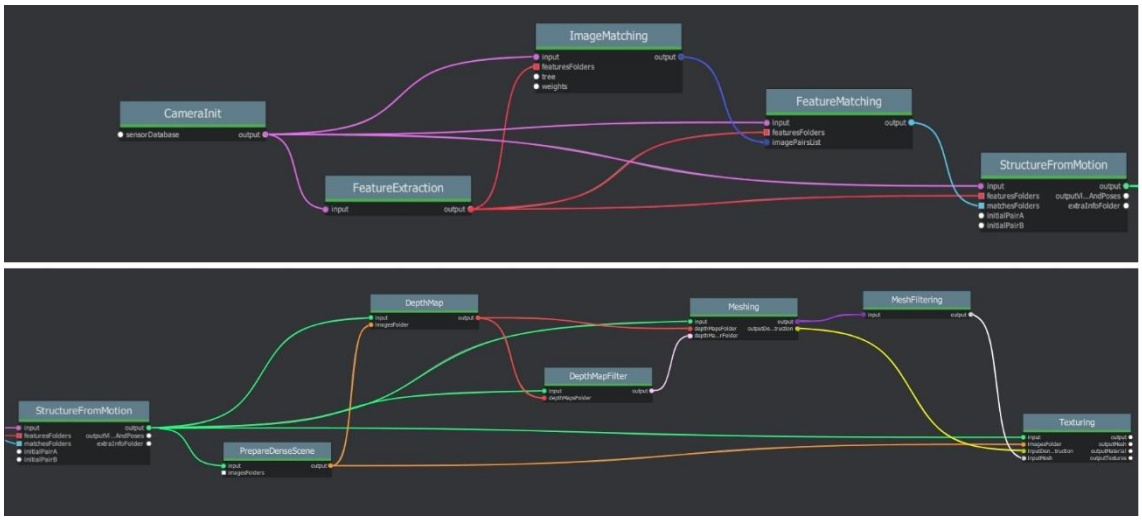


Figure 7. Main node tree used for this project, split in two and with the node connections on the origin nodes for a better interpretation. (Meshroom, 2020)

Meshroom uses Alembic files (.abc) by default for its Dense Point Clouds.

“Alembic is an interchangeable computer graphics file format developed by Sony Pictures Imageworks and Industrial Light & Magic. It was announced at SIGGRAPH 2011 and has been widely adopted across the industry by visual effects and animation professionals.

Its primary focus is the interchange of geometry (models) between different groups working on the same shots or same assets. Often different departments in the same company or different studios are working on the same projects. Alembic supports the common geometric representations used in the industry, including polygon meshes, subdivision surface, parametric curves, NURBS patches and particles. Alembic also has support for transform hierarchies and cameras. With the latest version comes initial support for materials and lights as well.” (Wikipedia, 2020)

While Alembic files were a good choice because they used a newer format and had the capability of storing a lot of different information between multiple applications, the software used in the next steps of the project either didn’t support .abc files at all, or the support was very limited at the moment. So, Alembic files needed to be converted to some other format.

A file format that could store similar information and had better software integration was the Polygon File Format, or Stanford Triangle Format, associated to .ply files. In order to convert the Meshing node Dense Point Cloud output automatically, since a manual conversion would defeat the whole purpose of this project, the ConvertSfMFormat node was used.

This node creates ‘abc’, ‘sfm’, ‘json’, ‘ply’ and ‘baf’ SfM File from SfMData files (the output of StructureFromMotion node) (Meshroom, 2020). The problem encountered here was that the conversion node was initially designed to work with Sparse Point Cloud representations, and it only saved the points associated to camera positions when used with Dense Point Clouds.

At this point, three alternatives existed in order to obtain the dense cloud data:

- **Convert .abc to .ply externally:**

Using Blender, a free and open source 3-D software, the alembic files could be imported as vertices. Then, using an addon called “Point Cloud Visualizer” (uhlik, 2020), the imported vertices could be converted to a point cloud in .ply format. This was the first method tried, since the main advantage was the ability to obtain .ply files for the dense point clouds without having to modify any software code. The two main downsides were that this method broke the fully automated pipeline, and that the converted .ply lost all the colour information, since the alembic points were converted to vertices.

- **Change the Meshing node output type:**

This method was based in a modification in the program code to add a drop-down menu in the meshing node to select the output file type. After seeing the results of the first proposed method, an alternative was searched inside Meshroom itself, to continue with the automated pipeline and to try and retain the point colour data.

By default, Meshroom was not able to do that because, while the software is written in Python code, the official installer releases came with all the files pre-packaged, so they couldn’t be modified.

Since Meshroom is an open-source software, the GitHub repository (AliceVision, 2020) was accessed to access the Issues section. There, a new post was created asking for help solving the issues when converting .abc files to .ply, since the only converted points in the output were the camera ones.

After some time, an answer from one of the developers was received, and after further discussing he provided a method to add an extension selection menu for the Meshing output node.

The implementation was first tested directly from the command line interface, without a proper menu, as seen in Figure 8:

Run the Meshing node (only, delete data if necessary) in the GUI with selected settings, then go to the CLI in the background and copy the parameters to a text file (select+enter). Then change abc extension to ply.
Open the Meshing folder and delete the files.

By running the Meshing node now from the CLI we will get our renamed file extension:

Start new CLI, navigate to ..\Meshroom-2019.2.0\aliceVision\bin
then type:

aliceVision_meshing.exe (paste your changed parameters set + enter to run)

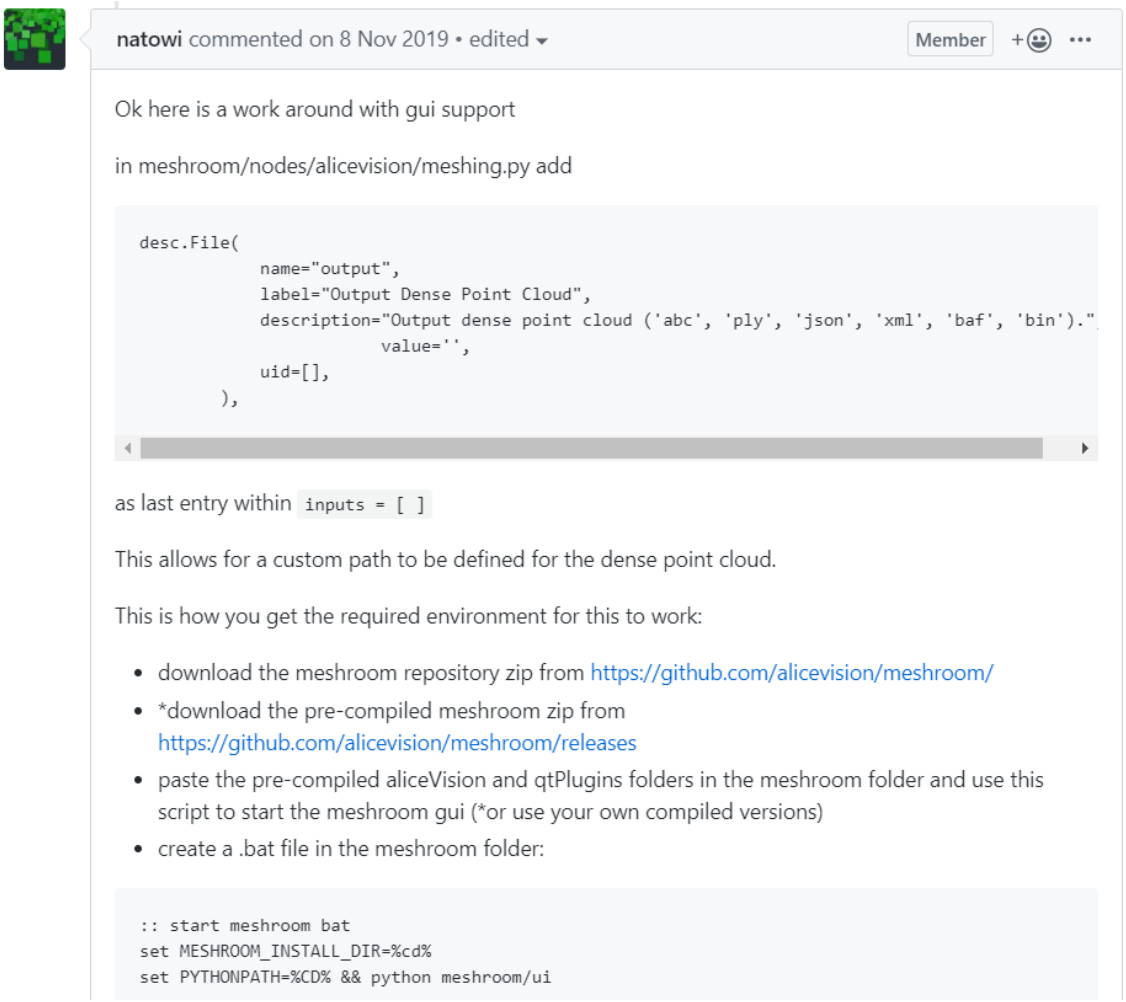
Example:

```
C:\Users\user>D:\Meshroom\Meshroom-2019.2.0-win64\Meshroom-
2019.2.0\aliceVision\bin\aliceVision_meshing.exe --input
"C:/Users/user/AppData/Local/Temp/MeshroomCache/StructureFromMotion/5893adbb9647185e705e57e98007c
459ebb29091/sfm.abc" --depthMapsFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMap/79bd8b4b35079d2c5a60be3b5302399952760ed5
" --depthMapsFilterFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMapFilter/47bbddab7c0b3ee1c10675095772b16e22
db6a01" --estimateSpaceFromSfM True --estimateSpaceMinObservations 3 --
estimateSpaceMinObservationAngle 10 --maxInputPoints 50000000 --maxPoints 5000000 --
maxPointsPerVoxel 1000000 --minStep 2 --partitioning singleBlock --repartition multiResolution --
angleFactor 15.0 --simFactor 15.0 --pixSizeMarginInitCoef 2.0 --pixSizeMarginFinalCoef 4.0 --
voteMarginFactor 4.0 --contributeMarginFactor 2.0 --simGaussianSizeInit 10.0 --simGaussianSize
10.1 --minAngleThreshold 1.0 --refineFuse True --addLandmarksToTheDensePointCloud False --
colorizeOutput True --saveRawDensePointCloud False --verboseLevel info --outputMesh
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/
mesh.obj" --output
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/
densePointCloud.ply"
```

Figure 8. Temporary solution provided by GitHub user "natowi" to obtain .ply dense point clouds.

This solution worked because Meshroom is only an interface used to send command instructions to the CLI (command line interface) that is opened alongside it when the software is launched. So, to get this method working, the desired parameters had to be set first in the GUI (graphic user interface) and then the node had to be executed. This way Meshroom generated the necessary command instruction, that could be copied and modified to get a .ply output instead of an .abc output. While this solution worked great, it could be tedious to use with a lot of files.

So, the next step was to add the .ply support directly to Meshroom. To do this, the developer provided some initial instructions to set up the development environment for Meshroom, as seen in Figure 9:



natowi commented on 8 Nov 2019 • edited — Member

Ok here is a work around with gui support

in meshroom/nodes/alicevision/meshing.py add

```
desc.File(
            name="output",
            label="Output Dense Point Cloud",
            description="Output dense point cloud ('abc', 'ply', 'json', 'xml', 'baf', 'bin')."
                        value='',
            uid=[],
        ),
```

as last entry within `inputs = [ ]`

This allows for a custom path to be defined for the dense point cloud.

This is how you get the required environment for this to work:

- download the meshroom repository zip from https://github.com/alicevision/meshroom/
- *download the pre-compiled meshroom zip from https://github.com/alicevision/meshroom/releases
- paste the pre-compiled aliceVision and qtPlugins folders in the meshroom folder and use this script to start the meshroom gui (*or use your own compiled versions)
- create a .bat file in the meshroom folder:

```
:: start meshroom bat
set MESHROOM_INSTALL_DIR=%cd%
set PYTHONPATH=%CD% && python meshroom/ui
```

Figure 9. Instructions provided by GitHub user "natowi" to implement .ply support in the GUI.

After some trouble setting up the required files, a folder dedicated to Meshroom development could be set up with the help of the developer, so the code could be modified as needed.

The software uses a different Python script for every implemented node, so this allowed to modify only the necessary one (Meshing node). Before implementing the provided solution, with a selection menu, my own code fix was tried. It consisted in changing the extension type in the output file for the dense point cloud from .abc to .ply directly from the Python script, as seen in Figure 10. This solution didn't allow for any output type change from the GUI but was good to obtain the results needed.

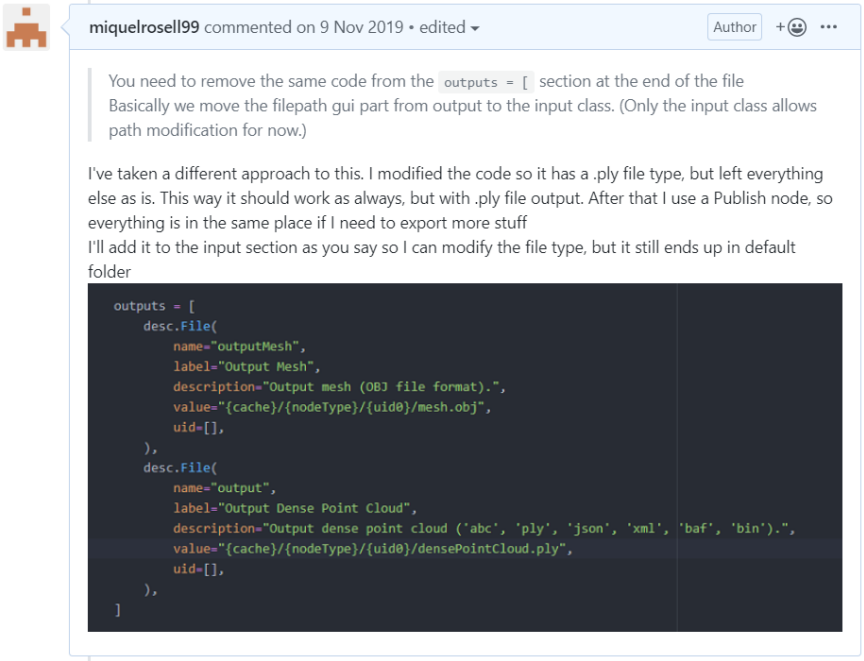


Figure 10. Personal approach for solving the format issue.

While this method was good enough, the menu was still thought to be very useful for future projects. So, using the commit feature in GitHub, where any code change can get assigned a unique code that other users can use to automate the implementation of new features, the menu was added. Each commit registers the code fragments that have been removed since the last modification, and the lines that have been added. In this case, commit 984e76e09ae54e45768083f7dc84de716fd94c33 was used, as seen in Figure 11:

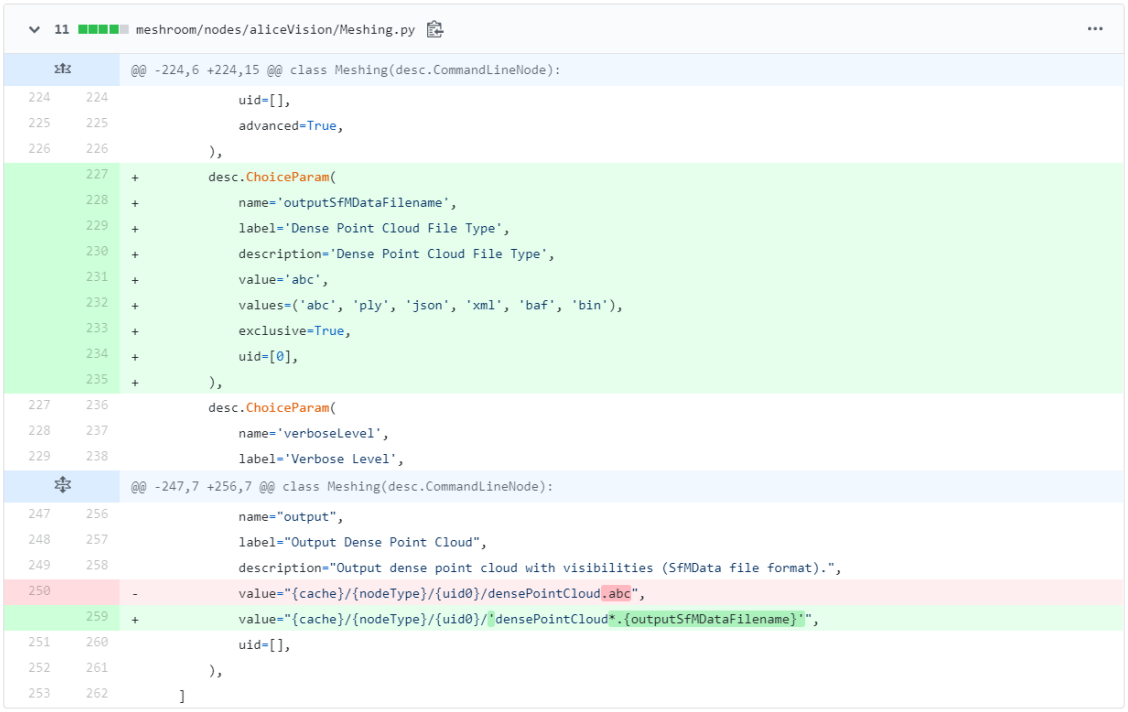


Figure 11. Code changes that enable a menu with file output types.

- **Using the new ConvertSfMFormat node:**

Having access to the source code allowed to update the software at any time. This opened the possibility of using newer node versions, with better feature implementations. The main developer of Meshroom, GitHub user “fabiencastan”, pointed out in the discussion that it iwas better to not modify the Meshroom node pipeline just to get a different file type, as seen in Figure 12:

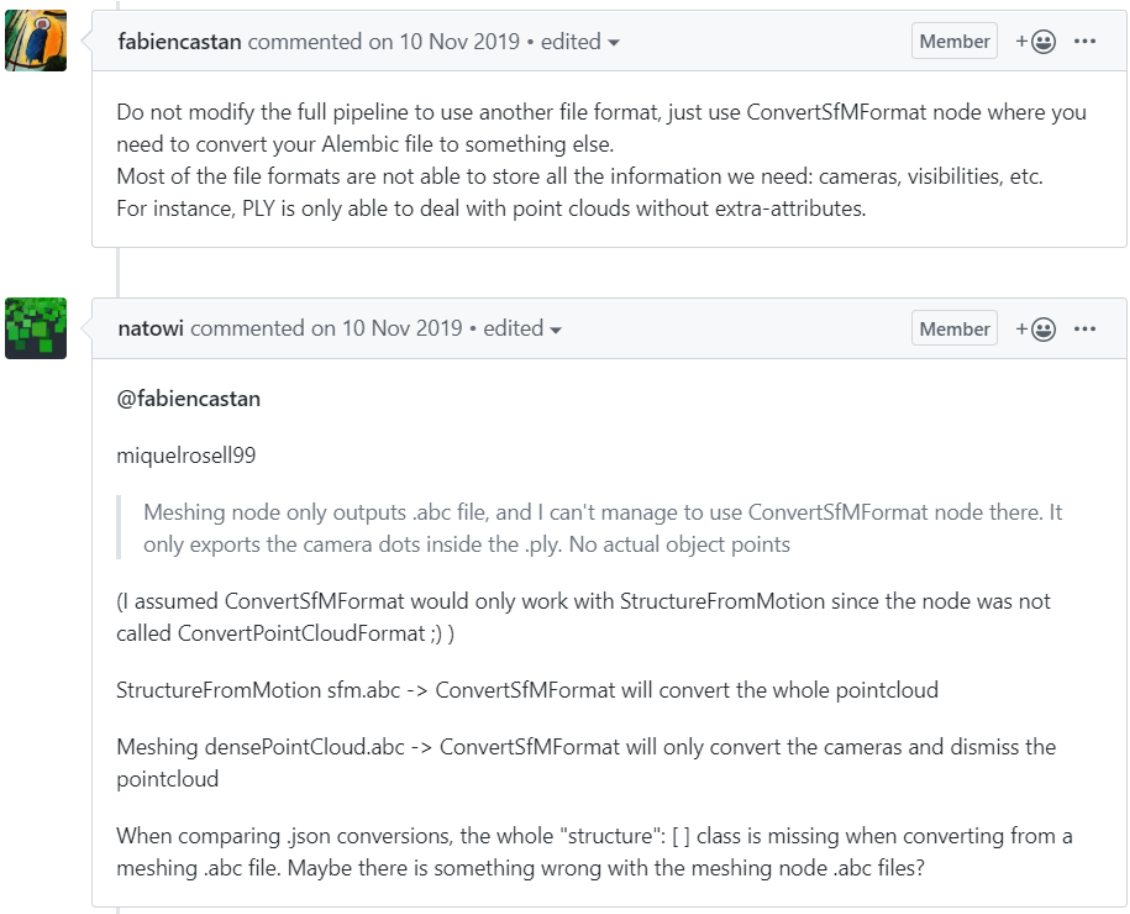


Figure 12. New method proposition by GitHub user and developer "faviencastan".

This argument was made because the previous method discussed to obtain .ply dense point clouds made the Meshing node incompatible with the posterior nodes in the pipeline, MeshFiltering and Texturing. This was not an issue for this project because they were not used at any point, but it meant that the code modifications could not be implemented in the main code public for everyone and had to be kept as a personal version.

So, instead of that solution, he suggested the usage of the new ConvertSfMFormat node implementation. This new version can convert dense point clouds to different formats if the correct describer type is selected. By default, the describer type is set to SIFT, and this gives the same output as the old version (a PLY file with only points associated to the camera positions). But when the describer type is switched from SIFT to Unknown, all the points get converted to PLY, so the output is the desired one.

This is the final adopted solution for this project, because the conversion took place in a separate step from the dense point cloud generation. This allowed to convert the .abc file to several file types at the same time without having to recalculate the project entirely.

So, by disabling the used algorithm under ConvertSfMFormat node settings and enabling "unknown describer type", the Dense Point Cloud could be obtained in .ply format with all its associated information (see Figure 2), as explained before.

Meshroom saves the data generated by each node into individual folders, that can be accessed via the Windows File Manager or by right clicking over the node of interest and selecting "Open Folder". This is a very powerful way of saving the data because it allows the user to access it at any time and from any software in order to process it even further.

The main disadvantage of this workflow is the creation of individual folder with random names for each project execution, that end up generating huge projects, in file quantity and in file size.

To avoid this, and to optimize the workflow automation, there is a node inside Meshroom called Publish. While other nodes only allow for one input in each connector, the Publish node lets the user connect its single input connector to as many node outputs as needed. Inside the node settings the user can then set a desired destination directory, inside the main project folder or in a different location, and all the plugged data will be **copied** there once the software reaches the Publish node in its sequential workflow. This means that the original processed files remain in the automatically generated folder, but they also get stored inside another folder with more ease of access. It also means that we can filter what kinds of outputs we want.

In this project the Dense Point Clouds were exported and then converted to .ply format using the appropriate node. Additionally, an option was enabled to also save the RAW dense point clouds without filtering. This option was used to compare the effect of the filtering in the results, since computing only the RAW point cloud took significantly less time when compared to the standard filtered option (minutes vs. hours).

2.2.3. Node explanation

This section is an overview of the different parameters contained in each one of the nodes used in this project. All the information is taken from the official Meshroom documentation. (Meshroom, 2020)

- **CameraInit**

"You can mix multiple cameras and focal lengths. The CameraInit will create groups of intrinsics based on the images metadata. It is still good to have multiple images with the same camera and same focal lengths as it adds constraints on the internal parameters of the cameras. But you can combine multiple groups of images, it will not decrease the quality of the final model.1

Note: In some cases, some image(s) have no serial number to identify the camera/lens device. This makes it impossible to correctly group the images by device if you have used multiple identical (same model) camera devices. The reconstruction will assume that only one device has been used, so if 2 images share the same focal length approximation, they will share the same

internal camera parameters. If you want to use multiple cameras, add a corresponding serial number to the EXIF data." (Meshroom, 2020)

Name	Description
Camera Intrinsics	(1 Element for each loaded image) - ID - Initial Focal Length: Initial Guess on the Focal Length - Focal Length: Known/Calibrated Focal Length - Camera Type: pinhole', 'radial1', 'radial3', 'brown', 'fisheye4' - #Make: Camera Make (not included in this build, commented out) - #Model: Camera Model - #Sensor Width: Camera Sensor Width - Width: Image - Width (0-10000) - Height: Image Height (0-10000) - Serial Number: Device Serial Number (camera and lens combined) - Principal Point: X (0-10000) Y(0-10000)- DistortionParams: Distortion Parameters - Locked(True/False): If the camera has been calibrated, the internal camera parameters (intrinsics) can be locked. It should improve robustness and speedup the reconstruction.
Sensor Database	Camera sensor width database path
Default Field of View	Empirical value for the field of view in degree 45° (0°-180°)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output SfMData File	…/cameraInit.sfm

Table 5. Available settings for the CameraInit node (Meshroom, 2020).

- **FeatureExtraction**

No explanation available in the documentation for this node.

- **ImageMatching**

This node has the function of finding images that are looking to the same areas of the scene, using the image retrieval techniques to find pictures that share some content without the computational cost of resolving all feature matches in detail. The objective is to simplify the image in a compact image descriptor, allowing to compute the distance between all image descriptors efficiently.

"One of the most common method to generate this image descriptor is the vocabulary tree approach. By passing all extracted features descriptors into it, it makes a classification by comparing their descriptors to the ones on each node of this tree. Each feature descriptor ends up in one leaf, which can be stored by a simple index: the index of this leaf in the tree. The image descriptor is then represented by this collection of used leaves indices.

It is now possible to see if different images share the same content by comparing these image descriptors." (AliceVision, 2020)

Name	Description
Image	SfMData file
Features Folders	Folder(s) containing the extracted features and descriptors
Tree	Input name for the vocabulary tree file ALICEVISION_VOCTREE
Weights	Input name for the weight file, if not provided the weights will be computed on the database built with the provided set

Minimal Number of Images	Minimal number of images to use the vocabulary tree. If we have less features than this threshold, we will compute all matching combinations
Max Descriptors	Limit the number of descriptors you load per image. Zero means no limit
Nb Matches	The number of matches to retrieve for each image (If 0 it will retrieve all the matches) 50 (0-1000)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output List File	Filepath to the output file with the list of selected image pairs

Table 6. Available settings for the ImageMatching node (Meshroom, 2020).

- **FeatureMatching**

"First, we perform photometric matches between the set of descriptors from the 2 input images. For each feature in image A, we obtain a list of candidate features in image B. As the descriptor space is not a linear and well-defined space, we cannot rely on absolute distance values to know if the match is valid or not (we can only have an absolute higher bound distance). To remove bad candidates, we assume that there's only one valid match in the other image. So, for each feature descriptor on the first image, we look for the 2 closest descriptors and we use a relative threshold between them. This assumption will kill features on repetitive structure but has proved to be a robust criterion [Lowe2004]. This provide a list of feature matching candidates based only on a photometric criterion. Find the 2 closest descriptors in the second image for each feature is computationally intensive with a brute force approach, but many optimized algorithms exist. The most common one is Approximate Nearest Neighbor, but there are alternatives like, Cascading Hashing.

Then, we use the features positions in the images to make a geometric filtering by using epipolar geometry in an outlier detection framework called RANSAC (RANdom SAmple Consensus). We randomly select a small set of feature correspondences and compute the fundamental (or essential) matrix, then we check the number of features that validates this model and iterate through the RANSAC framework." (AliceVision, 2020)

Name	**Description**
Input	SfMData file
Features Folders	Folder(s) containing the extracted features and descriptors
Image Pairs List	Path to a file which contains the list of image pairs to match
Describer Types	Describer types used to describe an image `**sift**'/ 'sift_float'/ 'sift_upright'/ 'akaze'/ 'akaze_liop'/ 'akaze_mldb'/ 'cctag3'/ 'cctag4'/ 'sift_ocv'/ 'akaze_ocv`
Photometric Matching Method	`For Scalar based regions descriptor ' * BRUTE_FORCE_L2: L2 Brute Force matching' ' * ANN_L2: L2 Approximate Nearest Neighbor matching ' * CASCADE_HASHING_L2: L2 Cascade Hashing matching ' * FAST_CASCADE_HASHING_L2: L2 Cascade Hashing with precomputed hashed regions (faster than CASCADE_HASHING_L2 but use more memory) 'For Binary based descriptor ' * BRUTE_FORCE_HAMMING: BruteForce Hamming matching'`

Geometric Estimator	Geometric estimator: `(acransac: A-Contrario Ransac // loransac: LO-Ransac (only available for fundamental_matrix model)`
Geometric Filter Type	Geometric validation method to filter features matches: `**fundamental_matrix** // essential_matrix // homography_matrix /// homography_growing // no_filtering'`
Distance Ratio	`Distance ratio to discard non meaningful matches 0.8 (0.0 - 1)`
Max Iteration	Maximum number of iterations allowed in ransac step 2048 `(1 - 20000)`
Max Matches	Maximum number of matches to keep `(0 - 10000)`
Save Putative Matches	putative matches (True/False)
Guided Matching	the found model to improve the pairwise correspondences (True/False)
Export Debug Files	debug files (svg/ dot) (True/False)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output Folder	Path to a folder in which computed matches will be stored

Table 7. Available settings for the FeatureMatching node (Meshroom, 2020).

- **StructureFromMotion**

This node uses the Incremental pipeline, which is a process of growing reconstruction. It first computes an initial reconstruction based on two views, that is then iteratively extended by adding new views from other imported images.

First, all feature matches between image pairs are fused into tracks, where each track is supposed to represent a point in space, visible from multiple cameras. At this step of the pipeline, each point can still contain many outliers. Incoherent tracks are removed during this match fusion.

Then, the best initial image pair is chosen by the incremental algorithm. This choice is critical to obtain a final reconstruction with a high quality. It should provide reliable geometric information and contain robust matches, maximizing the number of matches and the repartition of the corresponding features in each image, while at the same time having an angle between the cameras large enough to provide reliable geometric information.

With the two images selected, the node computes the fundamental matrix between them, and sets the first one as the origin of the coordinate system for the 3-D model. Once the pose of the first two cameras is known, corresponding 2D features can be triangulated into 3-D points.

On the next step, all the images that have enough associations with the features that are already reconstructed in 3-D are selected, using the next best views selection algorithm. Using these 2-D to 3-D associations, it performs the re-sectioning of each one of these new cameras, which uses a Perspective-n-Point algorithm (PnP) in a RANSAC framework to find the pose of the camera that can validate most of the feature associations. On each camera, and to refine the pose, the node performs a non-linear minimization.

Some tracks become visible by 2 or more resected cameras thanks to these new camera poses, so the node triangulates them. After that, in order to refine everything (extrinsic and intrinsic parameters of all cameras as well as the position of all 3-D points), it launches a Bundle Adjustment, and the results are filtered by removing all observations that have high reprojection error, or insufficient angles between different observations.

The node can then iterate like that, as we have triangulated new points, getting more image candidates for next best views selection. This keeps adding cameras and triangulating new 2-D features into 3-D points and removing 3-D points that became invalidated, until it can't localize new views.

"Many other approaches exist, like Global, Hierarchical or multi-stage." (AliceVision, 2020)

- **PrepareDenseScene**

Name	Description
Input	SfMData file
Verbose Level	['fatal', 'error', 'warning', 'info', 'debug', 'trace']
Output	MVS Configuration file (desc.Node.internalFolder + 'mvs.ini)

Table 8. Available settings for the PrepareDenseScene node (Meshroom, 2020).

- **DepthMap**

For each image, the node selects the N best/closest cameras around it. Based on the intersection of the optical axis with the pixels of the selected neighbouring cameras, it selects front-parallel planes, creating a volume with many depth candidates for each pixel. It then estimates the similarity for all of them, computed by re-projecting the Zero Mean Normalized Cross-Correlation (ZNCC) of a small patch in the main image onto the other camera, creating a volume of similarities.

Then node then accumulates similarities into this volume, that is very noisy, for each neighbouring image. To reduce the noise, it applies a filtering step along both (X and Y) axes, which accumulates local costs, drastically reducing the score of high values that are isolated. After that it selects the local minima, replacing the selected plane index with the depth value stored into a depth map, that has banding artifacts as it is based on the original selection of depth values. To get depth values with sub-pixel accuracy, a refine step is applied to all the generated maps.

All the depth maps can be computed in parallel, independently. (AliceVision, 2020)

Name	Description
`MVS Configuration File:`	SfMData file.
Images Folder	Use images from a specific folder instead of those specify in the SfMData file. Filename should be the image uid.
Downscale	Image downscale factor (1, 2, 4, 8, 16)
Min View Angle	Minimum angle between two views.(0.0 - 10.0)
Max View Angle	Maximum angle between two views. (10.0 - 120.0)
SGM: Nb Neighbour Cameras	Semi Global Matching: Number of neighbour cameras (1 - 100)

SGM: WSH: Semi Global Matching	Half-size of the patch used to compute the similarity (1 - 20)
SGM: GammaC	Semi Global Matching: GammaC Threshold (0 - 30)
SGM: GammaP	Semi Global Matching: GammaP Threshold (0 - 30)
Refine: Number of samples	(1 - 500)
Refine: Number of Depths	(1 - 100)
Refine: Number of Iterations	(1 - 500)
Refine: Nb Neighbour Cameras	Refine: Number of neighbour cameras. (1 - 20)
Refine: WSH	Refine: Half-size of the patch used to compute the similarity. (1 - 20)
Refine: Sigma	Refine: Sigma Threshold (0 - 30)
Refine: GammaC	Refine: GammaC Threshold. (0 - 30)
Refine: GammaP	Refine: GammaP threshold. (0 - 30)
Refine: Tc or Rc pixel size	Use minimum pixel size of neighbour cameras (Tc) or current camera pixel size (Rc)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output	Output folder for generated depth maps

Table 9. Available settings for the DepthMap node (Meshroom, 2020).

- **DepthMapFilter**

This filtering step is needed to ensure consistency between multiple cameras. “A compromise is chosen based on both similarity value and the number of coherent cameras to keep weakly supported surfaces without adding artifacts.” (AliceVision, 2020)

Name	**Description**
Input	SfMData file
Depth Map Folder	Input depth map folder
Number of Nearest Cameras	Number of nearest cameras used for filtering 10 (0 - 20)
Min Consistent Cameras	Min Number of Consistent Cameras 3 (0 - 10)
Min Consistent Cameras Bad Similarity	Min Number of Consistent Cameras for pixels with weak similarity value 4 (0 - 10)
Filtering Size in Pixels	Filtering size in Pixels (0 - 10)
Filtering Size in Pixels Bad Similarity	Filtering size in pixels (0 - 10)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output	Output folder for generated depth maps

Table 10. Available settings for the DepthMapFilter node (Meshroom, 2020).

- **Meshing**

This node fuses all the previously generated depth maps into a global octree. Compatible depth values are then merged into the octree cells to perform a 3-D Delaunay tetrahedralization. Then, a complex voting procedure is done to compute cell weights and facet weights. To optimally cut the volume, a Graph Cut Max-Flow is applied, which represents the extracted mesh surface. The node then filters bad cells on the surface and applies a Laplacian filtering on the whole mesh to remove localized artifacts. (AliceVision, 2020)

- **ConvertSfMFormat**

This node is used to convert point clouds to several file types, including .abc, .ply, .sfm and .baf.

Name	Description
Input	`SfMData file`
SfM File Format	SfM File Format (output file extension: abc', 'sfm', 'json', 'ply', 'baf)``
Describer Types	Describer types to keep. `'sift', 'sift_float', 'sift_upright', 'akaze', 'akaze_liop', 'akaze_mldb', 'cctag3', 'cctag4', 'sift_ocv', 'akaze_ocv'`
Image id	Image id
Image White List	image white list (uids or image paths).
Views	Export views
Intrinsics	Export intrinsics
Extrinsics	Export extrinsics
Structure	Export structure
Observations	Export observations
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output	Path to the output SfM Data file. (desc.Node.internalFolder + 'sfm.{fileExtension})
Refine Intrinsics	Enable/Disable camera intrinsics refinement for each localized image
Reprojection Error	Maximum reprojection error (in pixels) allowed for resectioning. If set to 0 it lets the ACRansac select an optimal value (0 - 10)
Use Localize Rig Naive	Enable/Disable the naive method for rig localization: naive method tries to localize each camera separately
Angular Threshold	The maximum angular threshold in degrees between feature bearing vector and 3-D point direction. Used only with the opengv method (0 - 10)
Voctree	[voctree] Filename for the vocabulary tree
Voctree Weights	[voctree] Filename for the vocabulary tree weights
Algorithm	[voctree] Algorithm type: {FirstBest, AllResults}``
Nb Image Match	[voctree] Number of images to retrieve in the database
Max Results	[voctree] For algorithm AllResults, it stops the image matching when this number of matched images is reached. If 0 it is ignored (0 - 100)
Matching Error	[voctree] Maximum matching error (in pixels) allowed for image matching with geometric verification. If set to 0 it lets the ACRansac select an optimal value (0 - 10)
N Nearest Key Frames	[cctag] Number of images to retrieve in database (0 - 50)
Output Alembic	Filename for the SfMData export file (where camera poses will be stored) desc.Node.internalFolder + 'trackedcameras.abc

Table 11. Available settings for the ConvertSfMFormat node (Meshroom, 2020).

- **Publish**

This node is used to automatically copy the files generated by the nodes connected to the input to a specific folder set as output. It can be very useful in order to have a centralized folder for final results.

Name	Description
Input Files	Input Files to publish
Output Folder	Folder to publish files to

Table 12. Available settings for the Publish node (Meshroom, 2020).

2.3. Device structure design

In order to be able to do successful 3-D scans, it's important to have a reliable scanning device. One portion of this reliability comes from the cameras, but another major role is played by the structure that will hold those cameras.

For this project an aluminum profile bar will be used as the base for the structure. This bar will have the devices attached at a certain height, that will allow some modifications in case of need.

The holder used to attach the smartphones to the aluminum profile was designed using the 3D design software PTC Creo Parametric, and the final design was 3-D printed in order to test it out. Due to the limitations introduced by Covid-19, the testing step could not be performed.

This holder needed to able to support different models of smartphone, in order to have the possibility to upgrade the devices in the future. Figure 13, Figure 14, ***¡Error! No se encuentra el origen de la referencia.*** and Figure 16 show the finished design:

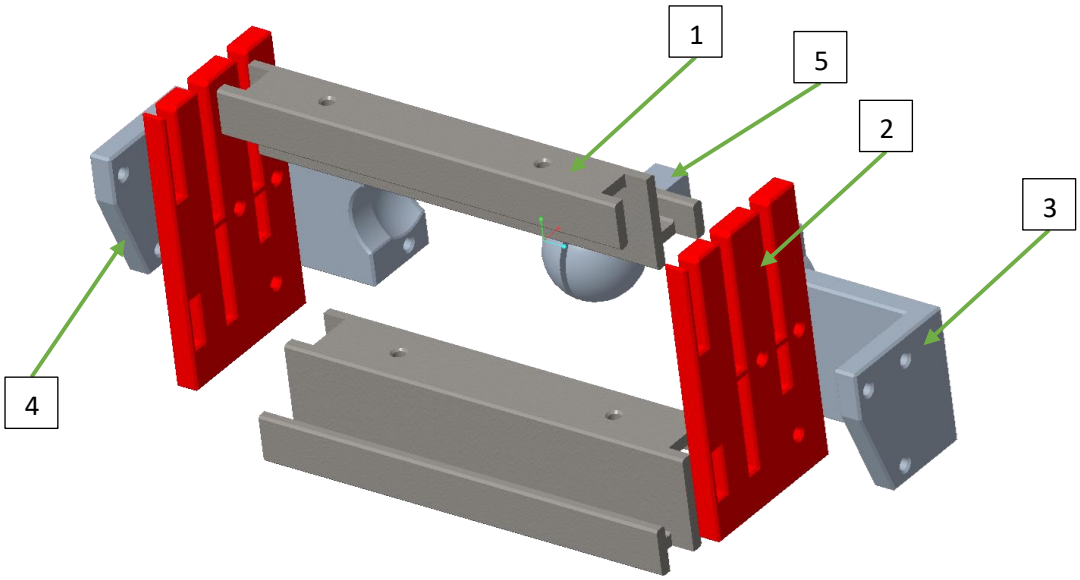


Figure 13 - Smartphone holder design in exploded view from the front

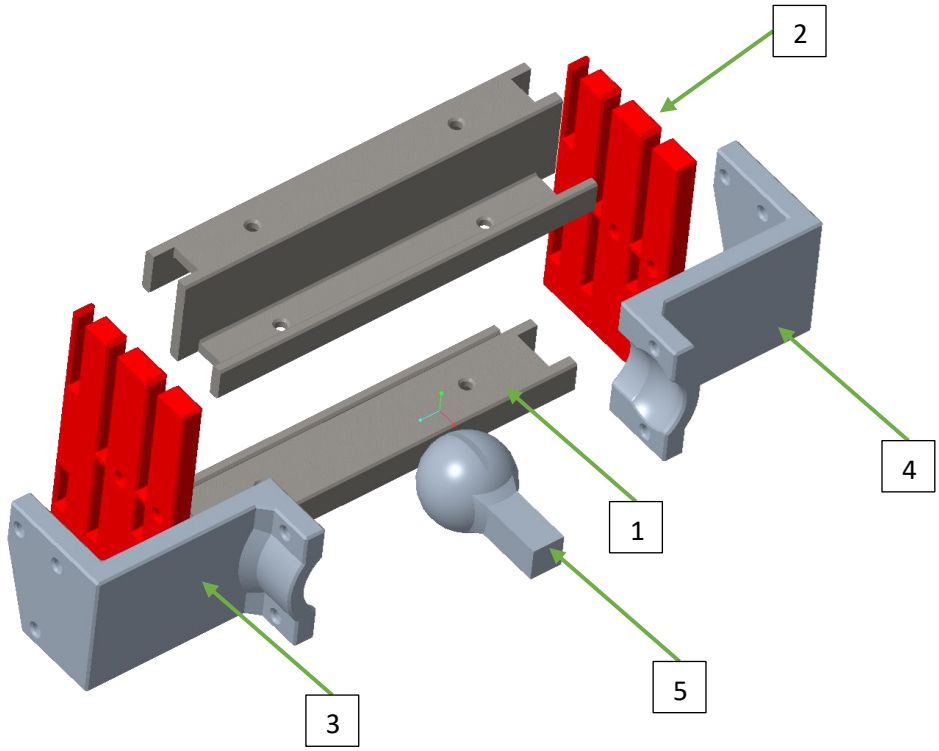


Figure 14 - Smartphone holder design in exploded view from the back

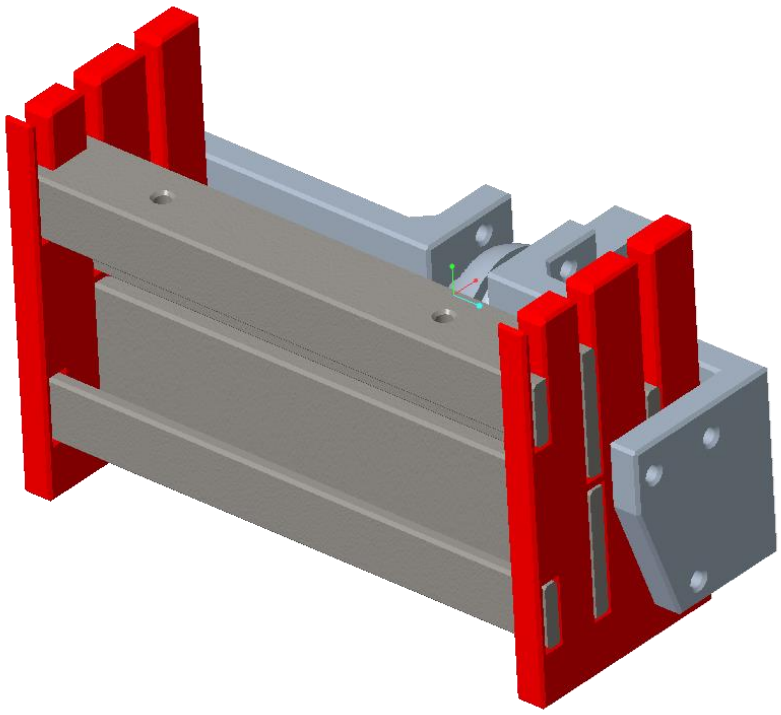


Figure 15 - Smartphone holder design from the front

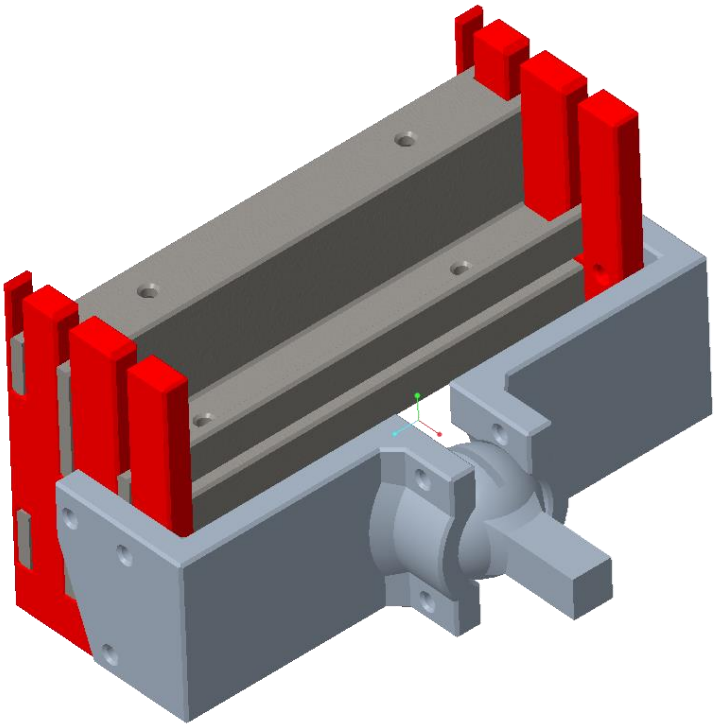


Figure 16 - Smartphone holder design from the back

This design used several parts to hold the devices:

- **Piece 1:**

This part is the one that holds the smartphone directly, by applying a clipping force from the bottom and the top. Two units are used, as seen in Figure 13 and Figure 14 in light brown.

- **Piece 2:**

This part acts as a guide for Piece 1. One unit is placed on the left, and another on the right. This prevents the smartphone from slipping through the side and keeps all the holder together. The bottom holes are used to block the movement of the Piece 1 located at the bottom, while the top rails act as horizontal constraints for the Piece 1 located at the top. Two units are used, as seen in Figure 13 and Figure 14 in red.

- **Piece 3:**

This part blocks the movement of Piece 2, so the integrity of the holder is guaranteed. The three fixing points ensure a secure locking without any kind of rotation allowed.

On the other side of the part there's the mechanism in charge of allowing or locking the rotation of the holder. As it can be seen in Figure 16, the holder has a ball pivot point, so a certain degree of rotation can be achieved from all axis. Once the desired rotation is set, it can be locked by tightening the two screws. It can be seen in Figure 13 and Figure 14 in grey.

- **Piece 4:**

This part is a symmetrical copy of Piece 3. Piece 3 is used for one side, and Piece 4 for the other. Both pieces are needed in order to lock Piece 2, and to create the ball pivot point. It can be seen in Figure 13 and Figure 14 in grey.

- **Piece 5:**

This part connects the holder with the aluminum profile bar, and acts as the pivot point for the holder thanks to the ball shape. It can be seen in Figure 13 and Figure 14 in grey.

2.4. Process automation

The most important thing to achieve a successful automation of the scanning method was to be able to automate and configure the image capturing.

When using smartphones, there were several ways to accomplish this objective:

- **Wireless controllers**

In the market there are lots of wireless devices that are used to control smartphone cameras remotely, for night photography, security, selfies… These devices could be used to trigger the camera shutter, but the main disadvantage was that each device could only control one smartphone at a time. Additionally, there was no easy way to automate the shutter button on the wireless devices.

- **Dedicated Android app**

A dedicated automation android app could be used to trigger the shutter button in the chosen camera app. “Intervalometer – Interval Timer for Time Lapse” is an app available on the Android Play Store that allows the users to configure a point in the device screen where the app will simulate a touch input following a configured interval. In the case of this project, it could be used to take an image every X second in order to follow the results of the calculations in Section 2.1.2.

With this control option, the shutter of all the devices could not be synced. This downside, though, was not significant for photogrammetry.

- **Headphone jack connection**

The headphone jack (or aux. jack) connection in smartphones could be used to trigger the shutter functionality inside most camera apps. This feature is usually used by selfie sticks, among others. By connecting the smartphones used in this project to a jack properly wired to an Arduino, and by controlling the current output in the pins, the shutter of the smartphone camera could be controlled to follow a specific interval. This control option also allowed the users to sync the shutters of all the used devices, by either using multiple pins in the Arduino, or by using a headphone jack splitter to connect the devices to the single output of the Arduino.

From the before-mentioned options, the dedicated android app was the one used for this project. While the headphone jack control opened more integration possibilities, due to the time limitation for this project the dedicated app was considered to be the most efficient in order to obtain the wanted results. Further investigation will be done for both control methods in order to accurately determine the pros and cons of each.

2.5. Test Proposal

In order to get a way to compare the obtained results of this photogrammetry scanning method to some reference, a 3-D simulation was used.

A real tree plantation could also be used for this purpose, but the only possibility to compare the results would be to scan the subjects with the method proposed in this final degree project and with another method known to have good results and then compare the output point clouds. This would introduce some error in the comparisons.

To solve this issue, simulated tests were performed using the free and open-source software Blender. This software lets the user create all sort of 3-D models, and it also allows the simulation of realistic cameras using real world parameters. This way the cameras present in the chosen device could be modelled for the scanning method, for then implementing those cameras in different 3-D scenes.

2.5.1. Simulating the cameras

For the practical application of the studied method, Xiaomi Mi 10 Lite was used, since this device had several sensors available that would allow for a more exhaustive testing. The mentioned practical application will not be discussed in this project due to the limitations as a consequence of Covid-19 but will be built and tested after the project presentation when the security measures can be met.

The characteristics of the used cameras can be found on several sources, and are the following:

- Primary camera: 48Mp 1/2-inch Sony IMX586 quad-sensor with 0.8µm pixels, f/1.75-aperture lens, 26mm-equivalent focal length.
- Telephoto camera: 12Mp 1/3.4-inch Samsung S5K3M5 sensor with 1.0µm pixels, f/2.2-aperture lens, 50mm-equivalent focal length.
- Super-wide-angle camera: 16Mp 1/3-inch Sony IMX481 sensor, f/2.2-aperture lens, 17mm-equivalent focal length.

The sensor size and other calculations can be seen in Section 2.1.2.

These cameras were then modelled in Blender, and the parameters can be seen in Figure 17, Figure 18 and Figure 19:

- **Main Camera:**

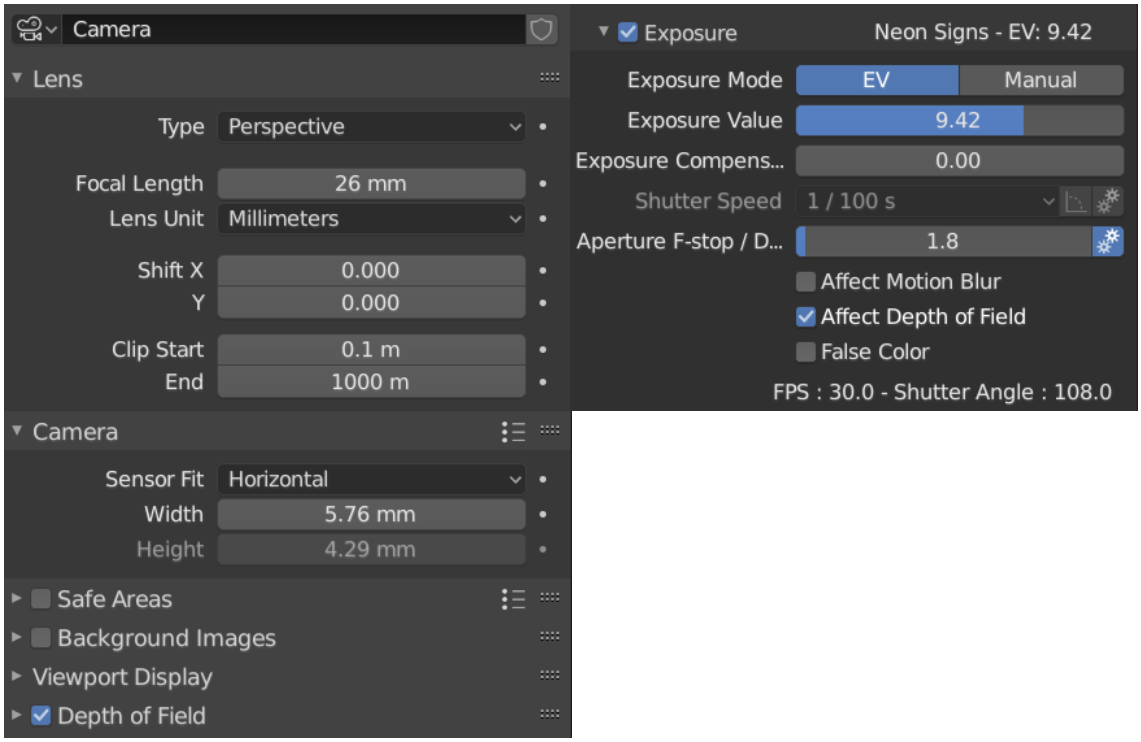


Figure 17. Main camera parameters in Blender

- **Telephoto Camera:**

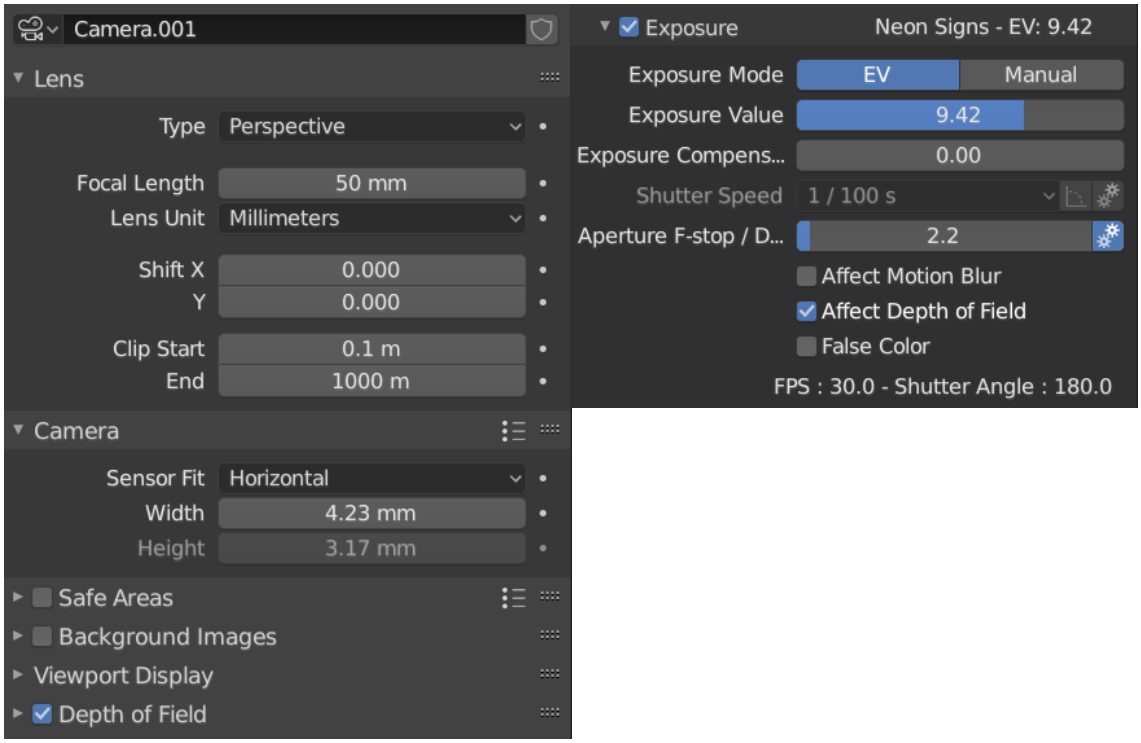


Figure 18. Telephoto camera parameters in Blender

- **Super-wide-angle Camera:**

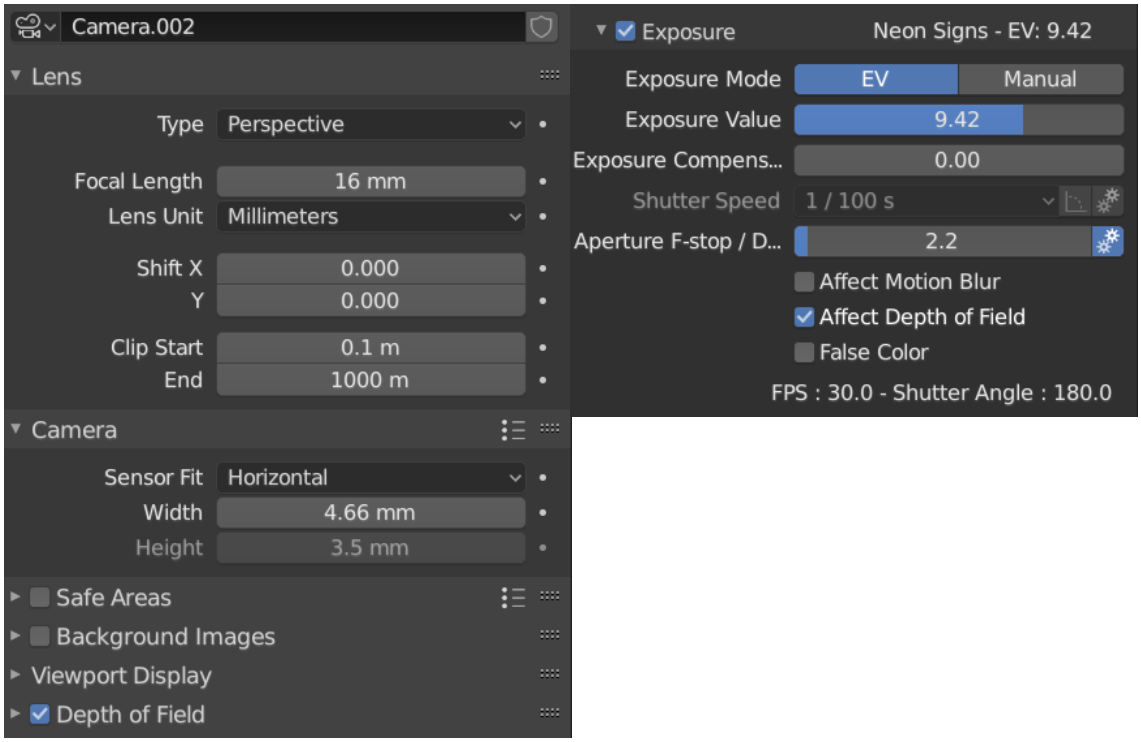


Figure 19. Super-wide-angle camera parameters in Blender

These cameras were positioned in the same layout of the designed support structure, as seen in Figure 20:

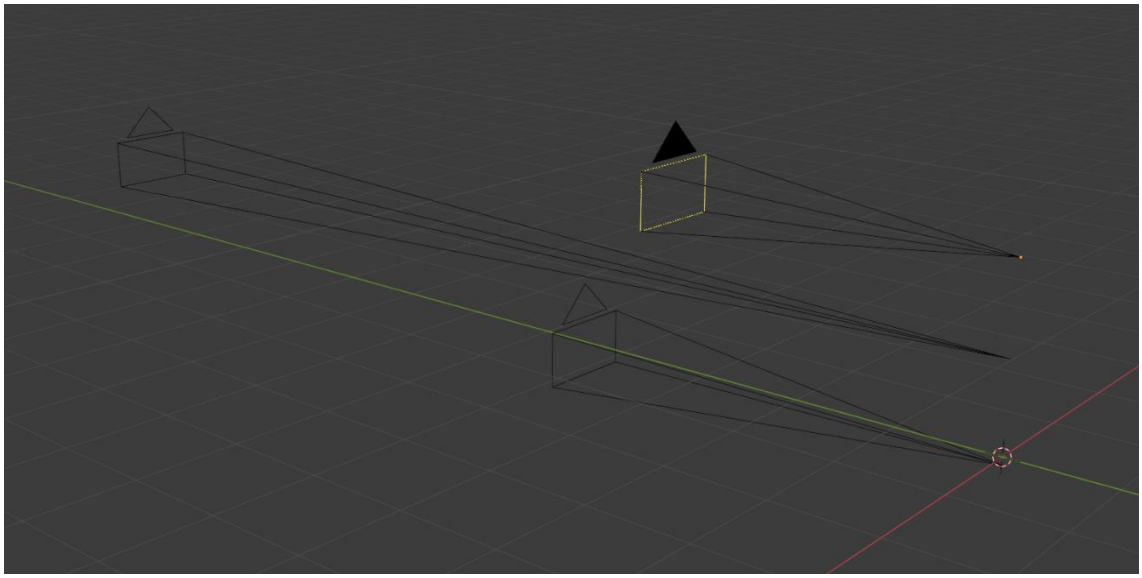


Figure 20. Camera distribution in the 3-D scene in Blender

The generated Blender .blend file was saved to a known location, so the camera setup could be easily imported inside different 3-D simulation scenes using the "Append" feature in the software, which is used to add assets from one file into another file, from meshes, to cameras, materials or entire scenes.

2.5.2. Building the simulation scenes

The goal of the simulations was to replicate the agricultural environments where the device would be mainly used. The main interest, then, was to be able to obtain realistic 3-D tree models to use as subjects for the scan. Blender gives the user the ability to obtain and install addons from a huge library, in order to expand the software's capabilities.

One of these addons, called The Grove 8 (The Grove 3D, 2020), adds the ability to generate realistic trees of different kinds procedurally. This means that as many tree models as needed could be obtained.

2.5.3. Performing the simulation

The idea of the simulation, then, was to make a camera movement as close as possible to the one the device would do in a real-life situation, then obtain the output images and import them inside Meshroom to perform all the needed calculations.

Inside Blender three options to export the obtained images are available:

- **Cycles render:**

Cycles is a physically based production renderer developed by the Blender project. It offers several features that allow the users to get realistic results. The images are processed using path-tracing algorithms, so the render times are a lot higher than on the other options, but it has the most realistic results. (Blender Foundation, 2020)

- **Eevee render:**

"Eevee is a newer physically based real-time renderer that has advanced features such as volumetrics, screen-space reflections and refractions, subsurface scattering, soft and contact shadows, depth of field, camera motion blur and bloom." (Blender Foundation, 2020). The main difference with the Cycles engine is that Eevee performs in real time, like what game engines work. This means that the rendering times are noticeably lower, but the results are not as physically accurate.

- **Viewport render:**

"Viewport rendering uses the 3D Viewport rendering for quick preview renders. This allows you to inspect your animatic (for object movements, alternate angles, etc.).

It can also be used to preview your animations – in the event your scene is too complex for your system to play back in real-time in the 3D View." (Blender Foundation, 2020)

The main advantage of this method is the speed at which the output is generated, while the main disadvantage is the complete lack of any physical simulation of the light.

For this project there was no need for a physically accurate render, since the only information needed were the different colours of the 3-D models. The ability to obtain renders with accurate depth of field was important, since the real cameras would have a certain background blur based on the aperture.

Based on these premises, the viewport render was the ideal choice for this project. A simulation with several hundred frames could be processed in a few minutes, compared to the hours it would take using physically based render engines. The viewport render also provided the option to process the depth of field depending on the camera characteristics, so all the requirements were met.

Meshroom has a processing mode called "Live mode", where the user can specify a folder where the images will be stored, and the software processes them in batches of a specified number of files as they get saved in the folder. Theoretically, this mode allows the user to see the photogrammetric reconstruction in real-time, so corrective adjustments can be performed on-the-go. One of the objectives was to use this mode in order to automate the process to the maximum, but after doing some testing it was seen that the performance was very poor, due to the large number of images, and their resolution. Therefore, this mode was disregarded for this project, although it could be used for smaller scans.

2.5.4. Test

2.5.4.1. High Dynamic Range Imaging (HDRI)

"High-dynamic-range imaging (HDRI) is a high dynamic range (HDR) technique used in imaging and photography to reproduce a greater dynamic range of luminosity than what is possible with standard digital imaging or photographic techniques. Standard techniques allow differentiation only within a certain range of brightness. Outside of this range, no features are visible because there is no differentiation in bright areas as everything appears just pure white, and there is no differentiation in darker areas as everything appears pure black.

HDR images can record and represent a greater range of luminance levels than can be achieved using more traditional methods, such as many real-world scenes containing very bright, direct sunlight to extreme shade, or very faint nebulae. This is often achieved by capturing and then combining several different, narrower range, exposures of the same subject matter. Non-HDR cameras take photographs with a limited exposure range, referred to as low dynamic range (LDR) or standard dynamic range (SDR), resulting in the loss of detail in highlights or shadows." (Wikipedia, 2020)

For this test, three image sets were taken:

- HDR+ Auto (or Live HDR): "Unlike earlier versions of High-dynamic-range (HDR) imaging, HDR+, also known as HDR+ on, uses computational photography techniques to achieve higher dynamic range. HDR+ takes continuous burst shots with short exposures. When the shutter is pressed the last 5–15 frames are analysed to pick the sharpest shots (using lucky imaging), which are selectively aligned and combined with image averaging. HDR+ also uses Semantic Segmentation to detect faces to brighten using synthetic fill flash and darken and denoise skies. HDR+ also reduces noise and improves colours, while

avoiding blowing out highlights and motion blur. HDR+ was introduced on the Nexus 6 and brought back to the Nexus 5." (Wikipedia, 2020)

- HDR+ Enhanced: "Unlike 'HDR+ Auto', 'HDR+ enhanced' mode does not use Zero Shutter Lag (ZSL). Like Night Sight, HDR+ enhanced features positive-shutter-lag (PSL): it captures images after the shutter is pressed. HDR+ enhanced is similar to HDR+ from the Nexus 5, Nexus 6, Nexus 5X and Nexus 6P. It is believed to use underexposed and overexposed frames like Smart HDR from Apple. HDR+ enhanced captures increase the dynamic range compared to HDR+ on. HDR+ enhanced on the Pixel 3 uses the learning-based AWB algorithm from Night Sight." (Wikipedia, 2020)
- HDR Disabled: Standard, or reduced, dynamic range photos.

For this test, 3 images were taken from every camera position. The first image used HDR+ Enhanced, the second image used HDR+ Auto, and the third image used no HDR (Standard Dynamic Range, or SDR).

The three different sets of images were processed using Meshroom, and the resulting dense point clouds were compared to analyse the behaviour of each method.

The main goal was to compare the image capture time, and to check if the higher dynamic range provided additional information in scenes with a lot of contrast compared to SDR. This would be the case if a scanning session took place under direct sunlight at 3 p.m. for example.

It is expected for HDR+ Enhanced to take longer, since it has to capture several images using different exposures. In comparison, SDR photos take only one shot at a specific exposure value. HDR+ Auto, on the other hand, uses ZSL (Zero Shutter Lag). This means that the photo is taken the instant the shutter is pressed, as explained earlier.

As for image information, it is expected for SDR to fall behind the other two method in darker areas due to the lack of multiple exposures. HDR+ Enhanced is expected to provide more information than HDR+ Auto under more challenging condition.

2.5.4.2. Compare 12 vs 48 MPx

The goal of this test was to compare how different resolutions affected the dense point cloud output. The hypothesis was that higher resolutions provide denser point cloud results at the cost of higher processing times.

For this purpose, two image sets were taken. Each image set used the same lens on the three devices in order to get the same resolution in all the images in the data set. The first image batch used a resolution of 6000x8000 pixels, and the second image batch used a resolution of 3000x4000 pixels. With these different outputs we were be able to analyse whether quadrupling the image resolution quadruples the number of points in the output, and if the processing time doubles when going from 3000x4000 to 6000x8000 pixels.

2.5.4.3. Compare same device orientation vs different orientation

For this test, two image sets were taken in order to see how different device orientation configurations could affect the output of a photogrammetry scan.

In the first one, all the devices had an orientation perpendicular to the ground, as seen in Figure 21:

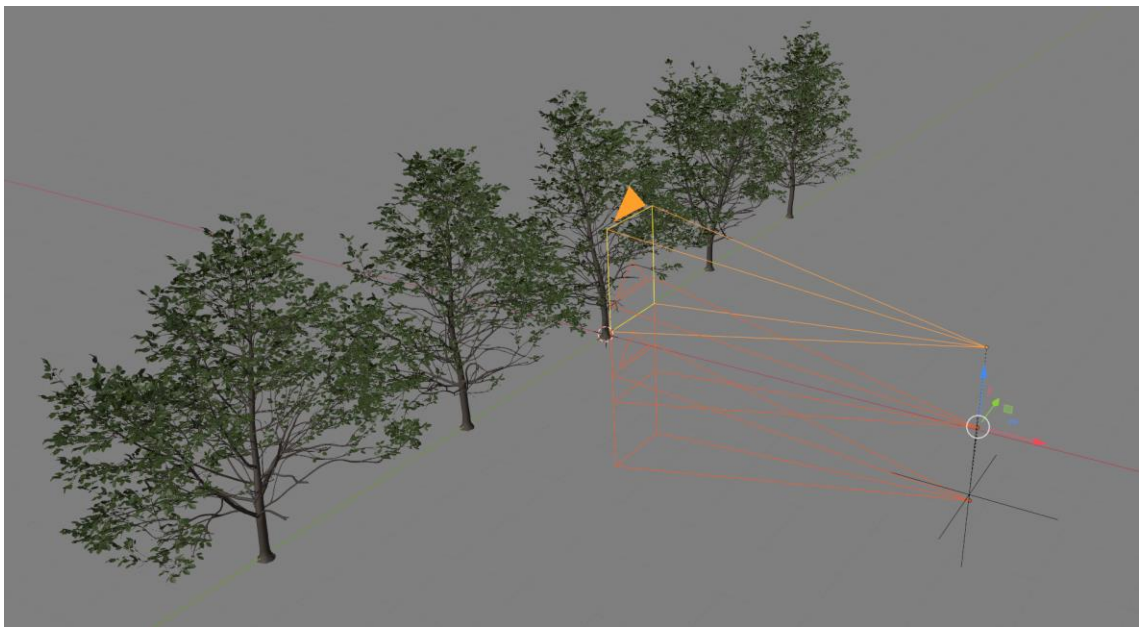


Figure 21 - All cameras with an orientation perpendicular to the ground, as seen in Blender viewport

For the second image set, the three devices formed a concave shape along the Y axis (vertical concave) around the scanning subject with an angle offset of 5 degree, as seen in Figure 22:

Figure 22 - All cameras with an orientation forming a concave shape along the Y axis, as seen in Blender viewport

It is expected that the dense point cloud resulting from the concave scan has a better depth accuracy, since the different orientation provide a different point of view of the central part of the scan. The main drawback of this method is expected to be a smaller field of view, since the top and bottom devices will be facing more towards the centre, and the edges of the scan subject will be cut from the scan.

2.5.4.4. Compare single image pass vs three image passes

The goal of this test is similar to the one in Section 2.5.4.3. The environments where the scanning device would be used are usually tight places, where there's not a lot of mobility. In these situations, performing scans around the subject may not be possible due to the space limitations, so only a parallel trajectory is feasible.

For this reason, this test aimed to try and incorporate different perspectives, or points of view, to the scan without changing the path followed by the scanning device.

Two image sets were taken. In the first one, the scanning device followed a parallel path and all the cameras were facing perpendicular to the subject (in this case, the line of trees, as seen in Figure 21). In the second one, two additional image sets were incorporated to the scan. Each image set had a slight positive or negative rotation offset compared to the original one, as seen in Figure 23. The test was done with an offset of 25 degrees first, and with an offset 15 degrees to compare the results. The 25-degree offset returned a cloud with missed references, so it was disregarded.

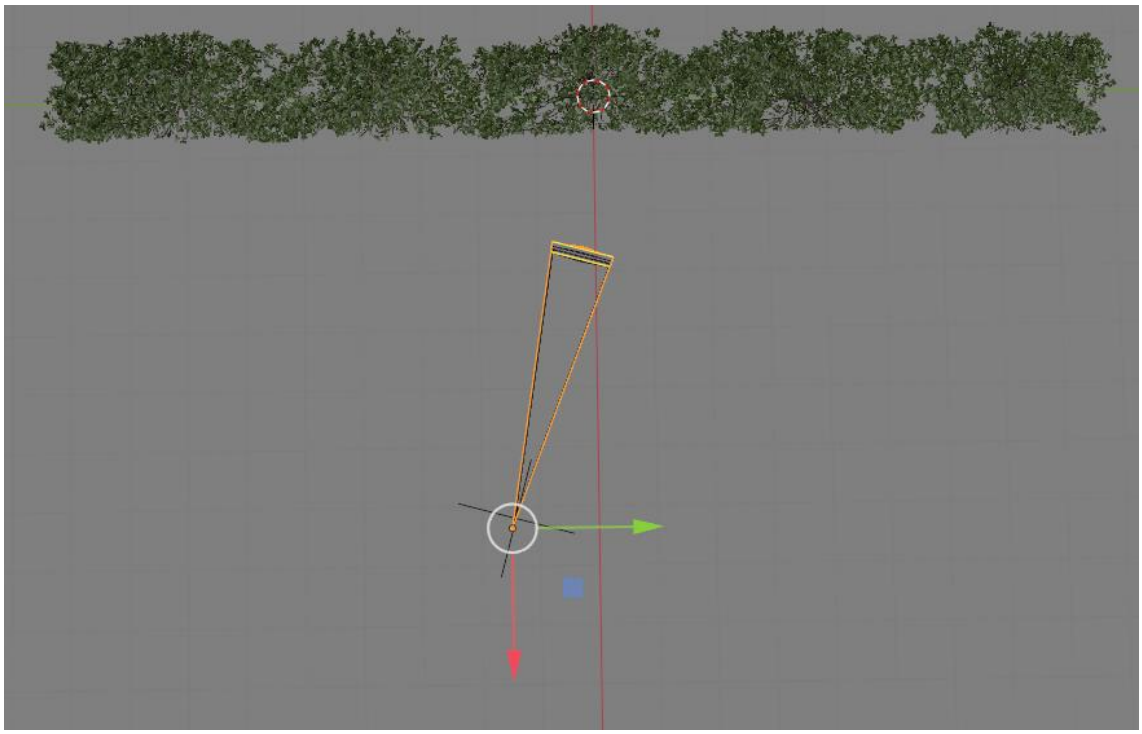


Figure 23 - Top view of one of the three passes performed in the test, with a rotation offset of 15 degrees along the Z axis. The direction of the followed path is marked by the green arrow along the Y axis.

The hypothesis is that these changes in the cameras rotation following the original path will provide a similar result to having all cameras perpendicular to the followed path, with the additional paths having a rotational offset compared to the original one, which would be the traditional way of performing a photogrammetry scan. Due to the space limitations when scanning a tree plantation, it can be difficult to perform multiple paths with different rotations, so being able so substitute those with a rotation of the capturing device would be very interesting.

All the additional images will result in an increase of the processing time, so this test aims to analyse if the extra time is worth the hypothetically improved results.

2.5.4.5. Compare video (no image metadata) vs photo (with metadata)

For this test, two image sets were taken:

- Photo source: When taking images directly from devices with a camera, these usually add some metadata regarding the sensor proprieties, the optics used, the GPS location, among other, embedding them inside the image file. This metadata can be used inside post processing software. In the case of Meshroom, it can process images from different sensors and optics inside the same project and correct the distortion that can appear when using wide angle lenses.
- Video source: In this case, the scan is performed by recording a video with the used device, and then extracting the frames from the video. This method has the major upside of being faster and easier than taking individual photos, so more information can be captured in a given time. The major drawback is the quality and sharpness of the taken images. Since a lot of images are being taken every second, the device can't apply any type of post-processing (HDR, for example) or keep focus in each shot. The motion blur is also more apparent.

The goal of this test was to scan the same subject using both methods and compare the obtained results.

2.5.5. Data Processing

In order to obtain error data for the performed scans, the first step was to obtain a point cloud from the tree meshes used for the simulation. This was done inside Cloud Compare by using the option "Sample points on a mesh", as seen in Figure 24:

Figure 24 - Option to sample points on a mesh in Cloud Compare, marked in orange

By using this option, a point cloud with a specific number of points could be obtained. In this case, 3.999.140 ≈ 4.000.000 points were sampled from the original mesh, as seen in Figure 25:

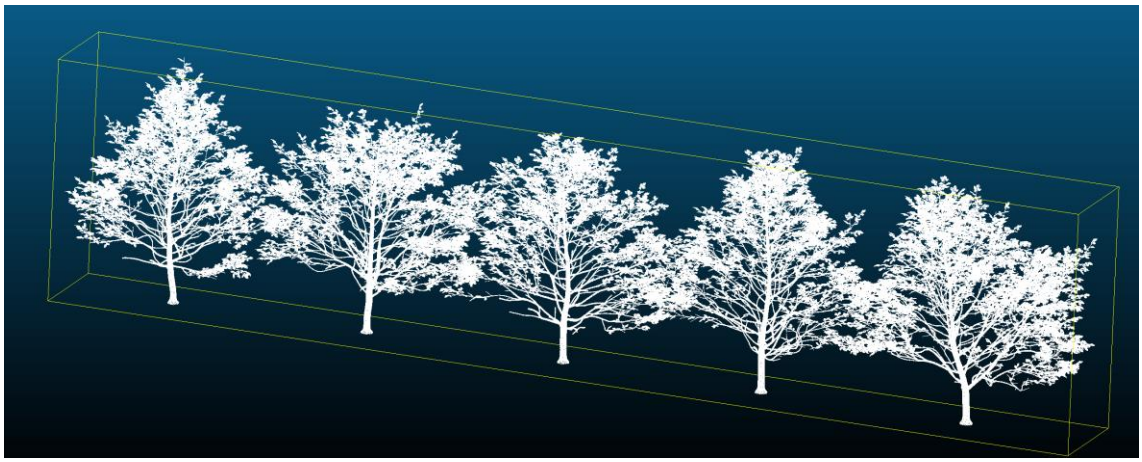


Figure 25 - Point cloud sampled from original mesh in Cloud Compare

Property	State/Value
CC Object	
Name	Model 5 arbres NÚVOL sampled 4M
Visible	☑
Show name (i...	☐
Colors	None
Box dimensions	X: 1.79957 Y: 17.8243 Z: 4.09307
Box center	X: 0.00192538 Y: -0.262969 Z: 1.99656
Info	Object ID: 17921 - Children: 0
Current Display	3D View 1
Cloud	
Points	3,999,140
Global shift	(0.00;0.00;0.00)
Global scale	1.000000
Point size	Default

Figure 26 - Sampled point cloud information

Having this point cloud allowed to compare the obtained clouds from the scan with the original one to check for missing zones easily.

The first step for processing the scan data was to properly align the obtained dense point clouds with the original mesh. For this, Blender was used to make a rough first approximation of the alignment. The roughly aligned point cloud was then imported into Cloud Compare and aligned to the original mesh finely, by using the option “Cloud Registration”, which finely registers already roughly aligned entities (in this case, a point cloud and a mesh), as seen in Figure 27.

Figure 27 - Option to register entities in Cloud Compare, marked in orange

The alignment was done with an RMS difference of 1.0e-5 meters, enabling the option to adjust the scale of the point cloud to the scale of the model.

The error analysis that was performed in a more in-depth manner comparing the obtained point clouds with the original mesh directly, to obtain distances from the points to the surface of the mesh in a perpendicular/normal direction, using the option “Compute cloud/mesh distance” inside Cloud Compare, as seen in Figure 28:

Figure 28 - Option to compute cloud/mesh distance in Cloud Compare, marked in orange

This option generates a scalar field with the distance corresponding to each point. The results can be displayed in a colour ramp, as seen in Figure 29:

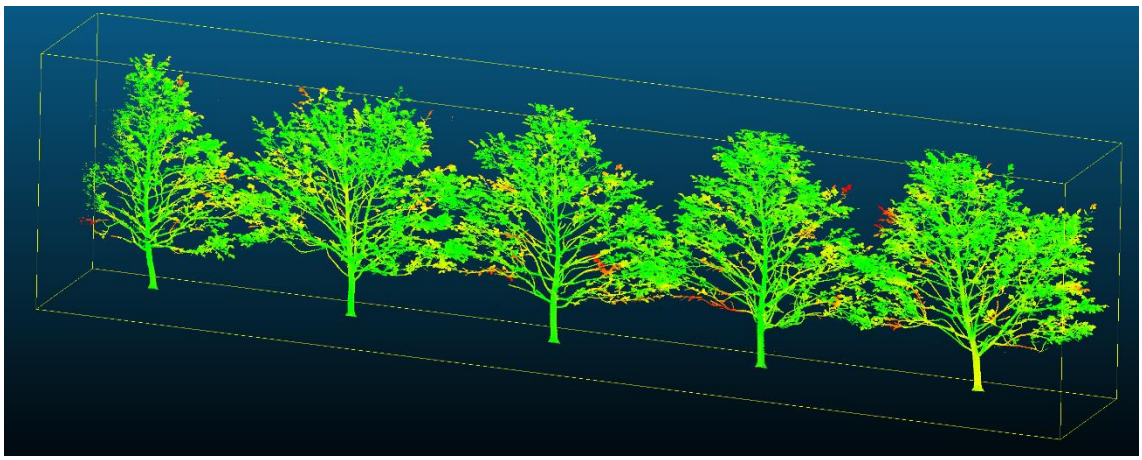


Figure 29 - Colour ramp representation of the scalar field "Error to mesh /m"

This comparison was done for the point clouds in both all configurations and processing types. The point clouds with their corresponding scalar fields were then saved as a text file with different columns for each variable. Figure 30 shows an example file:

```
//X Y Z R G B Error_to_mesh_/m Nx Ny Nz
0.5447 0.2322 2.7002 53 69 30 0.007 -0.999995 0.000981 -0.002944
-0.1998 -2.3426 1.5041 79 74 70 0.016 -0.999995
0.000981 -0.002944
-0.3869 5.6991 1.8322 69 82 54 0.004 -0.999995 0.000981 -0.002944
0.1588 -4.2303 1.0924 54 48 44 0.001 -0.999995 0.000981 -0.002944
-0.0285 -7.4994 3.2723 25 33 11 0.023 -0.999995
0.000981 -0.002944
-0.0379 4.7754 2.7851 23 26 16 0.001 -0.999995 0.000981 -0.002944
0.6578 7.9951 1.0328 38 38 30 0.015 -0.999995 0.000981 -0.002944
0.7772 -7.6017 2.4726 17 20 12 0.013 -0.999995 0.000981 -0.002944
-0.4322 5.6011 1.8932 50 63 37 0.005 -0.999995 0.000981 -0.002944
-0.0223 -7.7629 2.9403 41 53 23 0.009 -0.999995
0.000981 -0.002944
0.4684 -5.7841 2.6592 72 67 63 0.005 -0.999995 0.000981 -0.002944
-0.3980 7.9535 1.6827 25 26 18 0.017 -0.999995 0.000981 -0.002944
-0.6761 -5.8632 1.3405 66 59 54 0.002 -0.999995
0.000981 -0.002944
```

Figure 30 - Example layout for the output text file generated by Cloud Compare

These files were then imported inside JMP Pro 14, a predictive analysis software, in order to transform the text files into an organized table. Figure 31 shows an example of the obtained tables:

//X	Y	Z	R	G	B	Error_to_mesh_/ m
0,1403	3,5866	2,6853	26	28	17	0,002
0,3796	4,4215	1,852	59	73	44	0,002
0,6996	-3,214	1,7488	32	43	16	0,001
-0,0666	-5,6064	2,4165	49	62	34	0
0,4384	-7,8725	1,1927	71	66	62	0,005
-0,6083	-4,8406	1,3522	51	65	37	0,009
-0,4329	2,9526	1,5604	72	66	61	0,002
0,4687	-5,7824	2,7793	58	73	36	0,005

Figure 31 - Example layout for the table generated from the output text files

Each point cloud had its corresponding table. All the tables were concatenated, adding a column to indicate the camera configuration and a column to indicate the processing. The other variables were numeric. “Configuration” had three levels, and “Processing” had two levels.

A multifactorial lineal model was used to process the data obtained from the different test, in order to check the significance of the model and the significance of the factors and their interactions. With further processing, the normality and homoscedasticity assumptions were verified.

In the case that a violation occurred after the verification of the assumptions, a transformation for the response variable was considered, using the most appropriate transformation method. The obtained transformed data was also subjected to an assumption verification for the new model, for all the possible factor combinations.

In the case that the assumptions were correct, a median separation test was done with the Tukey-Kramer HSD test, since it is a common test used in the literature related to this topic and it is known to be a very conservative test. This means that it can be hard for it to find differences between the medians, but the ones that are found can be accepted without any doubts.

3. RESULTS

In this section, the results obtained from the methodology explained in Section 2 will be presented. Section 4 will further explain and discuss these results.

3.1. High Dynamic Range Imaging (HDRI)

Due to the consequences of Covid-19, this test could not be performed for this project.

3.2. Compare 12 vs 48 MPx

The results of this test followed the hypothesis proposed in Section 3. The obtained point clouds had 1.741.838 points when using 48 MPx images, and 891.723 points when using the 12 MPx dataset. Although the cloud point count was doubled, the processing time also increased by a considerable amount (close to double). Figure 32 shows the difference between both compared point clouds:

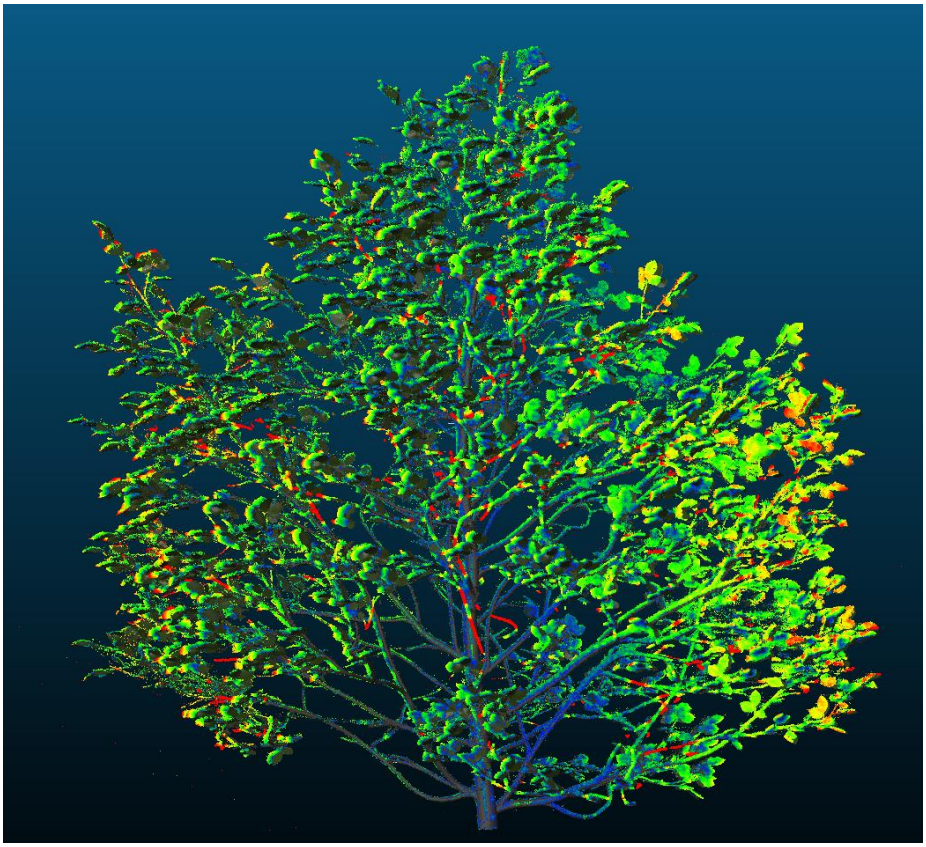


Figure 32 - Colour map of the point difference between a point cloud generated with 48 MPx pictures vs one generated from 12 MPx images.

3.3. Compare video (no image metadata) vs photo (with metadata)

Due to the consequences of Covid-19, this test could not be performed for this project.

The images used for all the tests had no sensor/optics metadata, so it was the theorical worst case scenario for the method. When this metadata is available in the files, Meshroom can process images from different sensors and optics inside the same project and correct the distortion that can appear when using wide angle lenses.

3.4. Compare same device orientation vs different orientation and single image pass vs three image passes

The distribution histograms for the error numerical variable are shown in Figure 33 to Figure 38 for each configuration and processing method combination, alongside the box diagram and a report of the quantiles and the statistics summary.

- **Same Orientation - Filtered**

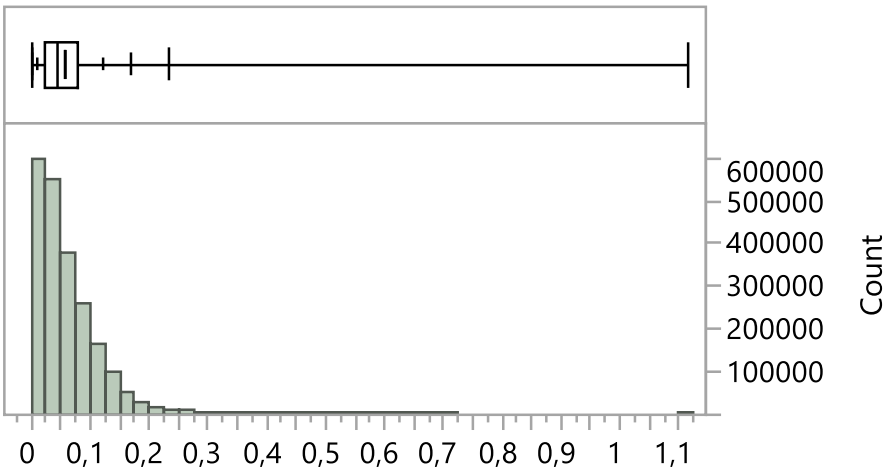


Figure 33 - Error to mesh in /m for Same Orientation - Filtered

Quantiles

100.0%	maximum	1,116
99.5%		0,234
97.5%		0,171
90.0%		0,12
75.0%	quartile	0,081
50.0%	median	0,045
25.0%	quartile	0,022
10.0%		0,01
2.5%		0,003
0.5%		0,001
0.0%	minimum	0

Summary Statistics

Mean	0,0569029
Std Dev	0,0460026
Std Err Mean	3,1648e-5
Upper 95% Mean	0,0569649
Lower 95% Mean	0,0568409
N	2112913
Skewness	1,3758914
Kurtosis	2,732206
CV	80,844037
Median	0,045
Mode	0,012
Range	1,116

- **Three Passes - Filtered**

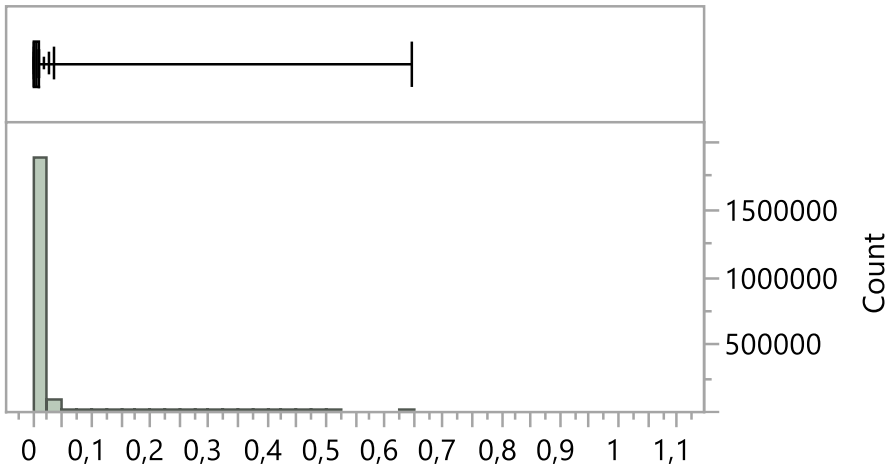


Figure 34 - Error to mesh in /m for Three Passes - Filtered

Quantiles

100.0%	maximum	0,646
99.5%		0,036
97.5%		0,026
90.0%		0,018
75.0%	quartile	0,012
50.0%	median	0,007
25.0%	quartile	0,003
10.0%		0,001
2.5%		0
0.5%		0
0.0%	minimum	0

Summary Statistics

Mean	0,008421
Std Dev	0,0073262
Std Err Mean	5,2623e-6
Upper 95% Mean	0,0084313
Lower 95% Mean	0,0084107
N	1938271
Skewness	3,8241826
Kurtosis	134,58058
CV	86,999704
Median	0,007
Mode	0,001
Range	0,646

- **Vertical Concave - Filtered**

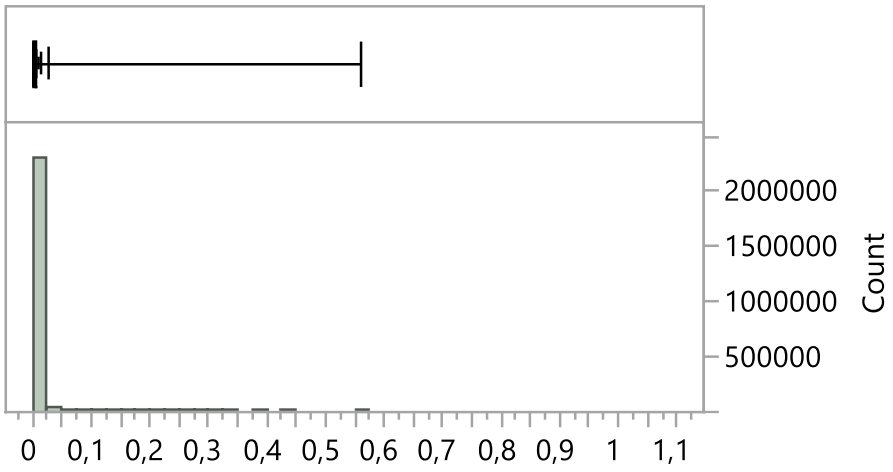


Figure 35 - Error to mesh in /m for Vertical Concave - Filtered

Quantiles

100.0%	maximum	0,561
99.5%		0,026
97.5%		0,016
90.0%		0,009
75.0%	quartile	0,006
50.0%	median	0,003
25.0%	quartile	0,001
10.0%		0,001
2.5%		0
0.5%		0
0.0%	minimum	0

Summary Statistics

Mean	0,004269
Std Dev	0,0046531
Std Err Mean	3,0687e-6
Upper 95% Mean	0,004275
Lower 95% Mean	0,004263
N	2299251
Skewness	6,1367996
Kurtosis	297,28096
CV	108,99833
Median	0,003
Mode	0,001
Range	0,561

- **Same Orientation RAW**

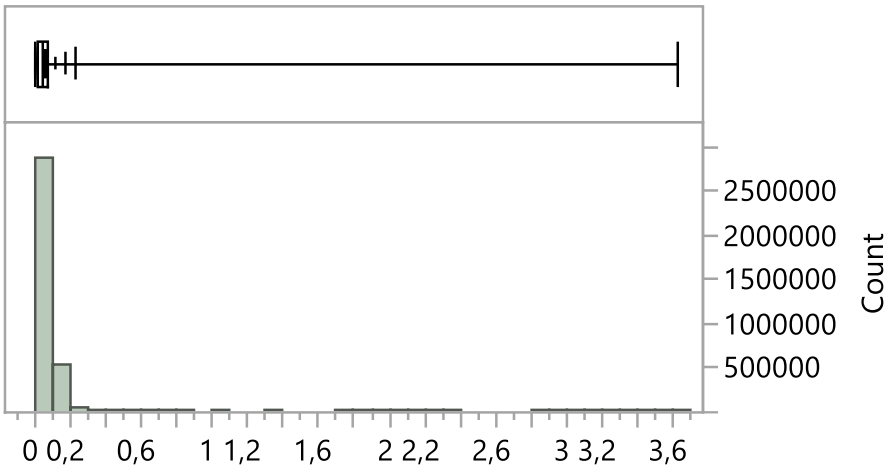


Figure 36 - Error to mesh in /m for Same Orientation - RAW

Quantiles

100.0%	maximum	3,621
99.5%		0,226
97.5%		0,167
90.0%		0,118
75.0%	quartile	0,079
50.0%	median	0,045
25.0%	quartile	0,022
10.0%		0,01
2.5%		0,003
0.5%		0,001
0.0%	minimum	0

Summary Statistics

Mean	0,0560561
Std Dev	0,0455378
Std Err Mean	2,4742e-5
Upper 95% Mean	0,0561046
Lower 95% Mean	0,0560076
N	3387346
Skewness	3,0922638
Kurtosis	119,82371
CV	81,236052
Median	0,045
Mode	0,017
Range	3,621

- **Three Passes - RAW**

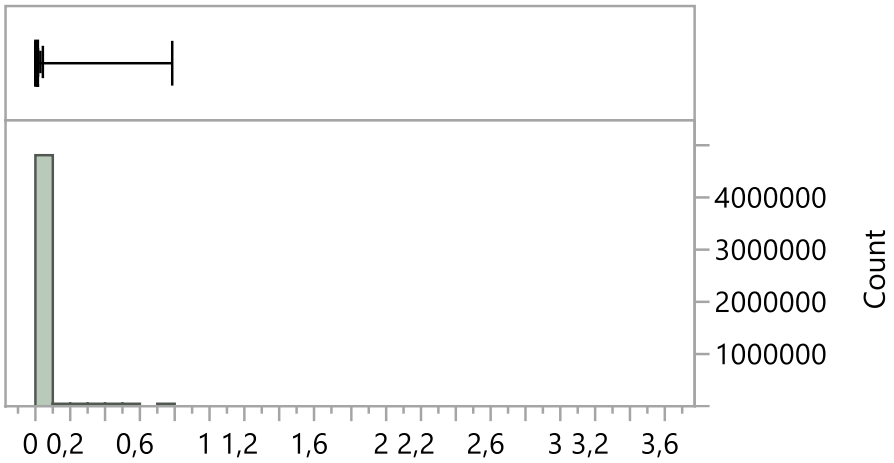


Figure 37 - Error to mesh in /m for Three Passes - RAW

Quantiles

100.0%	maximum	0,789
99.5%		0,046
97.5%		0,032
90.0%		0,021
75.0%	quartile	0,014
50.0%	median	0,007
25.0%	quartile	0,003
10.0%		0,001
2.5%		0
0.5%		0
0.0%	minimum	0

Summary Statistics

Mean	0,0097865
Std Dev	0,0090492
Std Err Mean	4,1477e-6
Upper 95% Mean	0,0097946
Lower 95% Mean	0,0097783
N	4760021
Skewness	3,0853619
Kurtosis	57,607031
CV	92,466346
Median	0,007
Mode	0,001
Range	0,789

- **Vertical Concave - RAW**

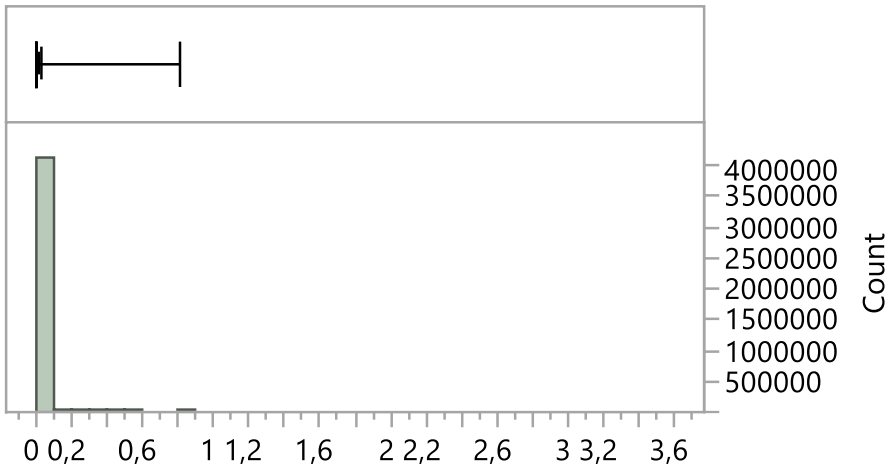


Figure 38 - Error to mesh in /m for Vertical Concave - RAW

Quantiles

100.0%	maximum	0,811
99.5%		0,033
97.5%		0,021
90.0%		0,012
75.0%	quartile	0,007
50.0%	median	0,004
25.0%	quartile	0,002
10.0%		0,001
2.5%		0
0.5%		0
0.0%	minimum	0

Summary Statistics

Mean	0,0052619
Std Dev	0,0058617
Std Err Mean	0,0000029
Upper 95% Mean	0,0052676
Lower 95% Mean	0,0052562
N	4087744
Skewness	5,2396245
Kurtosis	213,77861
CV	111,39853
Median	0,004
Mode	0,001
Range	0,811

The next Section discusses and analyses the obtained results.

4. DISCUSSION

4.1. Error analysis (ANOVA)

Analysis of the variance (ANOVA) in the error (Error), based on the fixed factors “Configuration” (Same Orientation, Three Passes and Vertical Concave) and “Processing” (RAW and Filtered). For this, a linear bifactorial model with interaction (fixed effects) must be proposed. If the model presents significance, the best combination between “Configuration” and “Processing” can be established. For an accurate interpretation of the results, the dependent variable (Error) must be transformed according to its distribution.

4.1.1. Model

First, a lineal model has to be adjusted for the available data. The model needs to have two factors, with interactions between them. As it can be seen in Figure 39, the model is significative (Analysis of Variance with Prob > F <0,0001), and only the "Configuration" factor is significative (Source with PValue = 0,0000 (< 0,05 and Effect Tests with Prob > F < 0,0001), although further assumption validation is required. Neither "Processing" nor the interaction is significative.

Response Error_to_mesh/m
Effect Summary

Source	LogWorth	PValue
FACTOR 1 - CONFIGURATION	490,674	0,00000
FACTOR 2 - PROCESSING	0,406	0,39222
FACTOR 1 - CONFIGURATION*FACTOR 2 - PROCESSING	0,150	0,70747

Summary of Fit

RSquare	0,440377
RSquare Adj	0,439658
Root Mean Square Error	0,026161
Mean of Response	0,023249
Observations (or Sum Wgts)	3900

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	2,0971545	0,419431	612,8512
Error	3894	2,6650252	0,000684	Prob > F
C. Total	3899	4,7621797		<,0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0,0232492	0,000419	55,50	<,0001*
FACTOR 1 - CONFIGURATION[Same Orientation]	0,0327069	0,000592	55,21	<,0001*
FACTOR 1 - CONFIGURATION[Three Passes]	-0,014375	0,000592	-24,26	<,0001*
FACTOR 2 - PROCESSING[Filtered]	-0,000358	0,000419	-0,86	0,3922
FACTOR 1 - CONFIGURATION[Same Orientation]*FACTOR 2 - PROCESSING[Filtered]	0,0004685	0,000592	0,79	0,4291
FACTOR 1 - CONFIGURATION[Three Passes]*FACTOR 2 - PROCESSING[Filtered]	-0,000367	0,000592	-0,62	0,5357

Effect Tests

Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
FACTOR 1 - CONFIGURATION	2	2	2,0961797	1531,416	<,0001*
FACTOR 2 - PROCESSING	1	1	0,0005011	0,7322	0,3922
FACTOR 1 - CONFIGURATION*FACTOR 2 - PROCESSING	2	2	0,0004737	0,3461	0,7075

Figure 39 - Response for Error_to_mesh/m

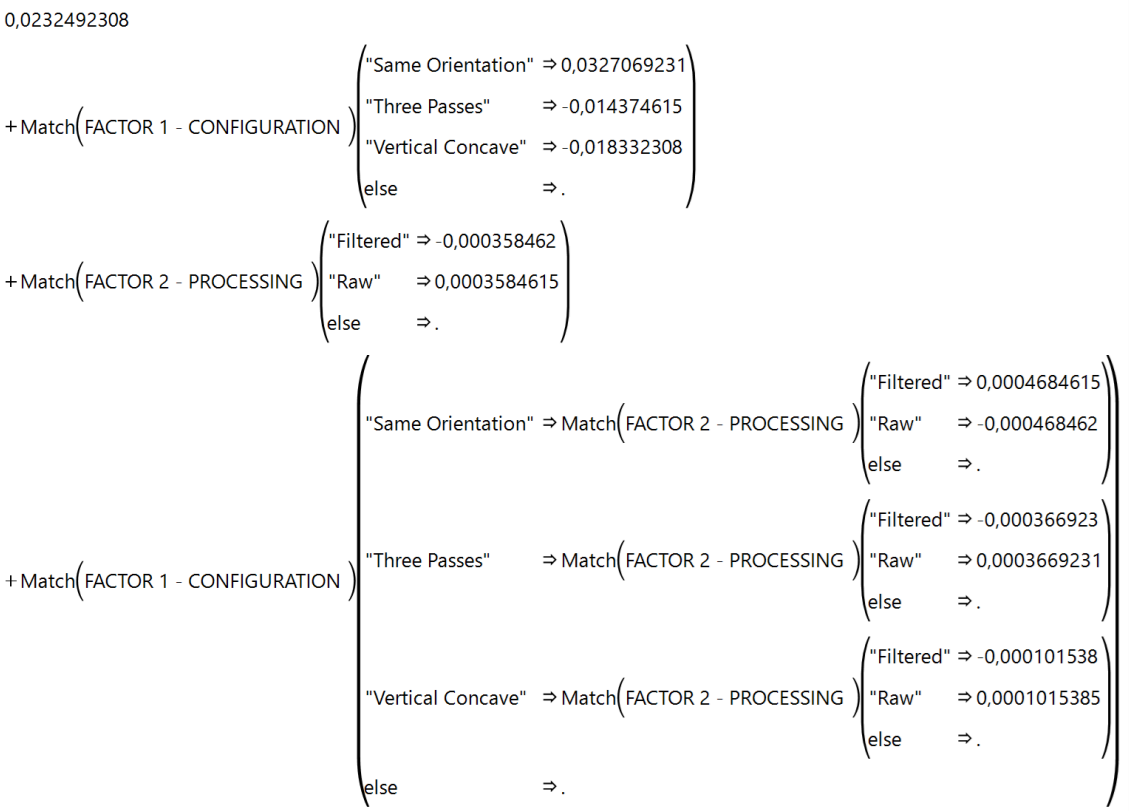


Figure 40 – Prediction expression for Error_to_mesh/m

4.1.2. Assumption validation (Model 1)

- **Homoscedasticity**

To check the homoscedasticity, a comparison between the variance of all 6 possible combinations between factors has to be done. The diagram in Figure 41 shows a clearly big difference in the variance between interaction groups. Additionally, O'Brien, Brown-Forsythe, Levene and Bartlett tests have all a Prob > F smaller than 0,0001.

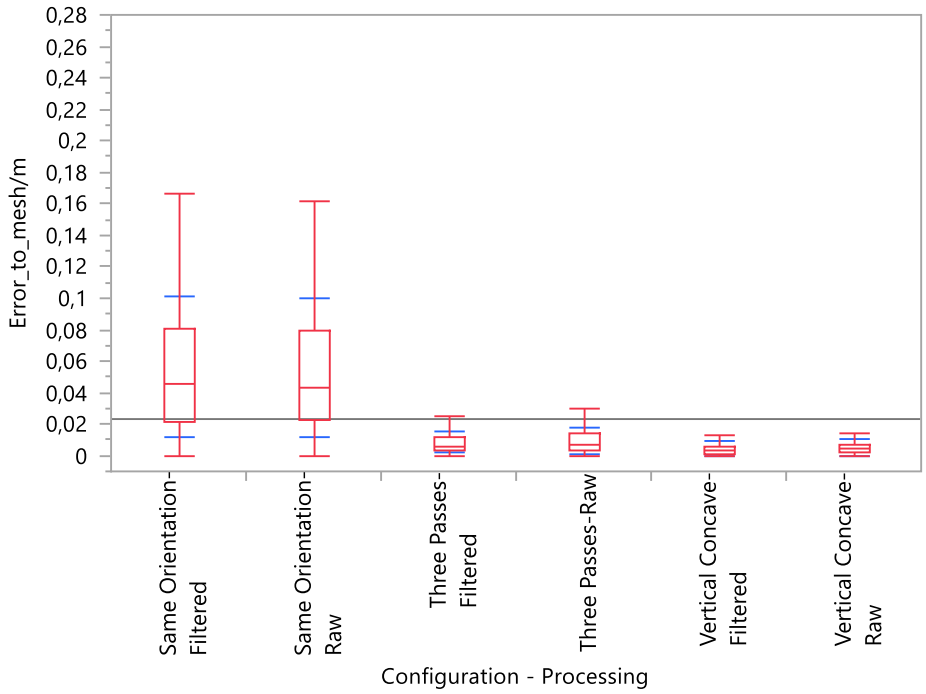


Figure 41 - Oneway analysis of Error_to_mesh/m by Configuration - Processing

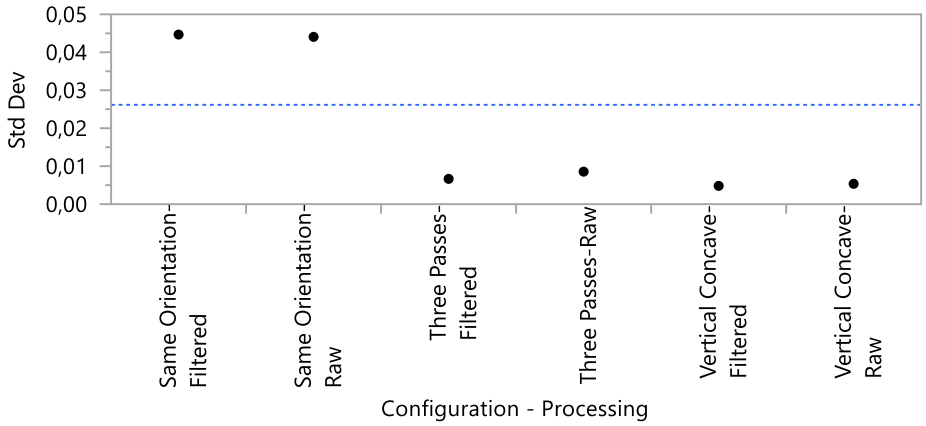


Figure 42 - Test to check if the variances are equal

Level	Count	Std Dev	MeanAbsDif to Mean	MeanAbsDif to Median
Same Orientation-Filtered	650	0,0446695	0,0346983	0,0336200
Same Orientation-Raw	650	0,0440640	0,0347415	0,0334708
Three Passes-Filtered	650	0,0066577	0,0052641	0,0050846
Three Passes-Raw	650	0,0085536	0,0065643	0,0062923
Vertical Concave-Filtered	650	0,0048145	0,0032997	0,0030262
Vertical Concave-Raw	650	0,0053556	0,0038652	0,0036446

Test	F Ratio	DFNum	DFDen	Prob > F
O'Brien[.5]	129,2115	5	3894	<,0001*
Brown-Forsythe	430,8457	5	3894	<,0001*
Levene	588,9028	5	3894	<,0001*
Bartlett	1221,6651	5	.	<,0001*

Welch's Test

Welch Anova testing Means Equal, allowing Std Devs Not Equal

F Ratio	DFNum	DFDen	Prob > F
376,7027	5	1767,4	<,0001*

4.1.3. Transformation of the response variable (Error)

Due to the results in the normality test, we need to transform the response variable. Since there are pieces of data with a value of 0, the Box-Cox transformation must be discarded, and Johnson Sb transformation will be used in order to find the normality.

With Johnson's transformation, the results can be normalized to a certain degree, as shown in the plots in Figure 43, Figure 44, Figure 45, Figure 46, Figure 47 and Figure 48.

- **Distributions FACTOR 1 - CONFIGURATION=Same Orientation, FACTOR 2 - PROCESSING=Filtered**

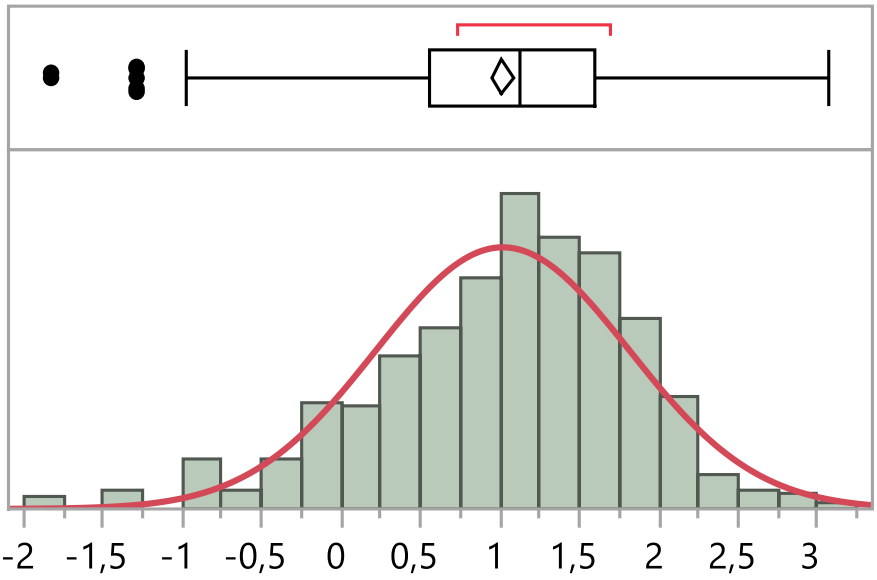


Figure 43 - Johnson Sb distribution [Error to mesh in /m] - Normal (1,01692,0,79773)

Quantiles

100.0%	maximum	3,0695967028
99.5%		2,8017475426
97.5%		2,354147631
90.0%		1,9498152292
75.0%	quartile	1,5965964605
50.0%	median	1,1204805656
25.0%	quartile	0,5543599107
10.0%		-0,078331719
2.5%		-0,764786308
0.5%		-1,689383196
0.0%	minimum	-1,826677018

Summary Statistics

Mean	1,0169227
Std Dev	0,7977348
Std Err Mean	0,0312897
Upper 95% Mean	1,078364
Lower 95% Mean	0,9554813
N	650

Fitted Normal
Parameter Estimates

Type	Parameter	Estimate	Lower 95%	Upper 95%
Location	μ	1,0169227	0,9554813	1,078364
Dispersion	σ	0,7977348	0,7566	0,8436351

Measure	
-2*LogLikelihood	1549,8473
AICc	1553,8658
BIC	1562,8012

Goodness-of-Fit Test

Shapiro-Wilk W Test

W	Prob<W
0,976972	<,0001*

Note: Ho = The data is from the Normal distribution. Small p-values reject Ho.

- **Distributions FACTOR 1 - CONFIGURATION=Same Orientation, FACTOR 2 - PROCESSING=Raw**

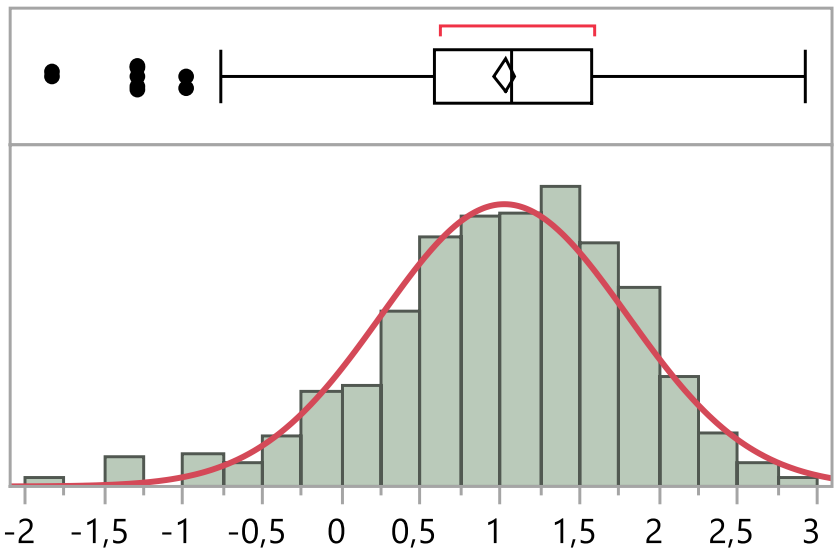


Figure 44 - Johnson Sb distribution [Error to mesh in /m] - Normal (1,02454,0,77531)

Quantiles

100.0%	maximum	2,9173419659
99.5%		2,6999538288
97.5%		2,3243952412
90.0%		1,9421538179
75.0%	quartile	1,5743886734
50.0%	median	1,0666535662
25.0%	quartile	0,5872506636
10.0%		-0,006664338
2.5%		-0,764786308
0.5%		-1,288269874
0.0%	minimum	-1,826677018

Summary Statistics

Mean	1,0245385
Std Dev	0,7753093
Std Err Mean	0,0304101
Upper 95% Mean	1,0842527
Lower 95% Mean	0,9648244
N	650

Fitted Normal
Parameter Estimates

Type	Parameter	Estimate	Lower 95%	Upper 95%
Location	μ	1,0245385	0,9648244	1,0842527
Dispersion	σ	0,7753093	0,7353309	0,8199193

Measure	
-2*LogLikelihood	1512,7789
AICc	1516,7975
BIC	1525,7329

Goodness-of-Fit Test

Shapiro-Wilk W Test

W	Prob<W
0,979493	<,0001*

Note: Ho = The data is from the Normal distribution. Small p-values reject Ho.

- **Distributions FACTOR 1 - CONFIGURATION=Three Passes, FACTOR 2 - PROCESSING=Filtered**

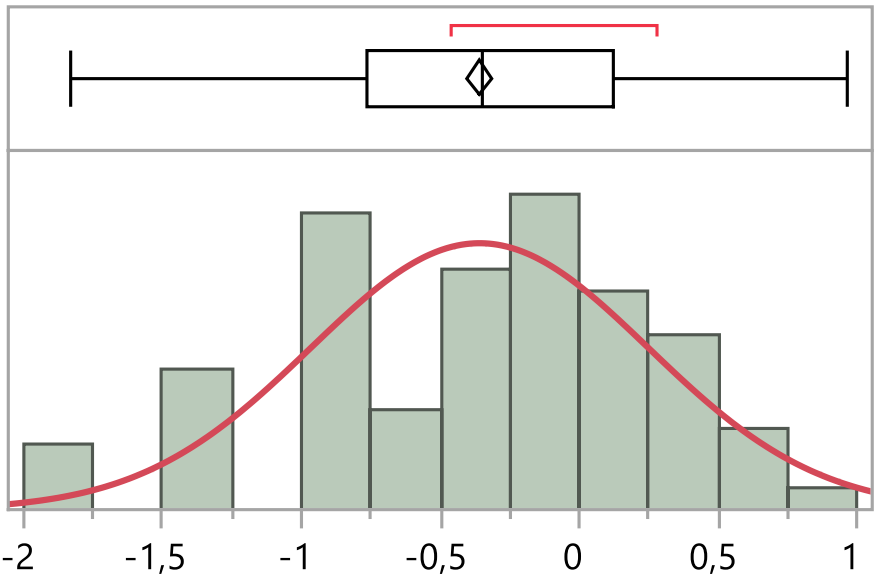


Figure 45 - Johnson Sb distribution [Error to mesh in /m] - Normal (-0,3585,0,60789)

Quantiles

100.0%	maximum	0,9692258068
99.5%		0,8537799662
97.5%		0,6493258801
90.0%		0,3662653582
75.0%	quartile	0,1192762696
50.0%	median	-0,3457484
25.0%	quartile	-0,764786308
10.0%		-1,288269874
2.5%		-1,826677018
0.5%		-1,826677018
0.0%	minimum	-1,826677018

Summary Statistics

Mean	-0,358529
Std Dev	0,6078934
Std Err Mean	0,0238435
Upper 95% Mean	-0,311709
Lower 95% Mean	-0,405349
N	650

Fitted Normal
Parameter Estimates

Type	Parameter	Estimate	Lower 95%	Upper 95%
Location	μ	-0,358529	-0,405349	-0,311709
Dispersion	σ	0,6078934	0,5765477	0,6428706

Measure	
-2*LogLikelihood	1196,5377
AICc	1200,5562
BIC	1209,4916

Goodness-of-Fit Test

Shapiro-Wilk W Test

W	Prob<W
0,972703	<,0001*

- **Distributions FACTOR 1 - CONFIGURATION=Three Passes, FACTOR 2 - PROCESSING=Raw**

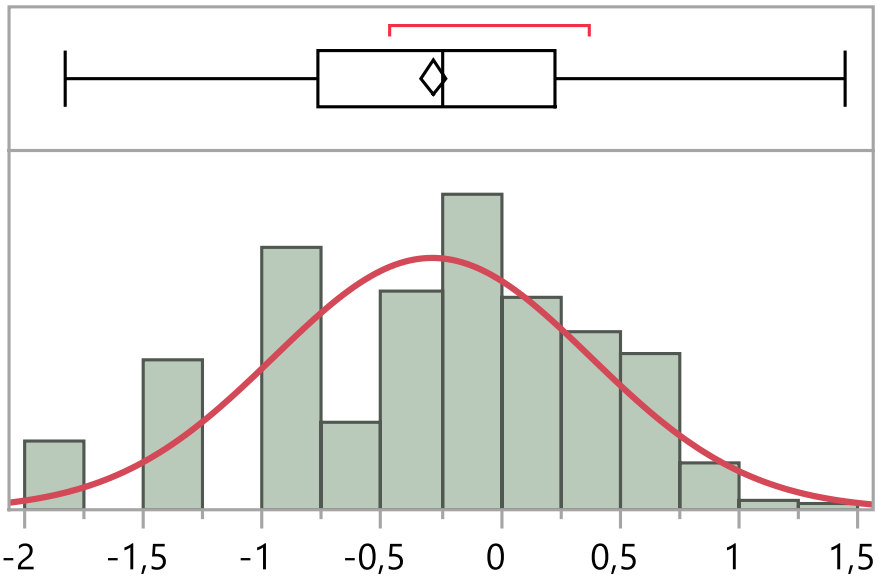


Figure 46 - Johnson Sb distribution [Error to mesh in /m] – Normal (-0,2875,0,66457)

Quantiles

100.0%	maximum	1,4438405812
99.5%		1,0418094875
97.5%		0,7868608645
90.0%		0,5543599107
75.0%	quartile	0,2275743136
50.0%	median	-0,245740323
25.0%	quartile	-0,764786308
10.0%		-1,288269874
2.5%		-1,826677018
0.5%		-1,826677018
0.0%	minimum	-1,826677018

Summary Statistics

Mean	-0,287452
Std Dev	0,6645733
Std Err Mean	0,0260667
Upper 95% Mean	-0,236266
Lower 95% Mean	-0,338637
N	650

Fitted Normal
Parameter Estimates

Type	Parameter	Estimate	Lower 95%	Upper 95%
Location	μ	-0,287452	-0,338637	-0,236266
Dispersion	σ	0,6645733	0,6303049	0,7028117

Measure	
-2*LogLikelihood	1312,4269
AICc	1316,4455
BIC	1325,3809

Goodness-of-Fit Test

Shapiro-Wilk W Test

W	Prob<W
0,974996	<,0001*

- **Distributions FACTOR 1 - CONFIGURATION=Vertical Concave, FACTOR 2 - PROCESSING=Filtered**

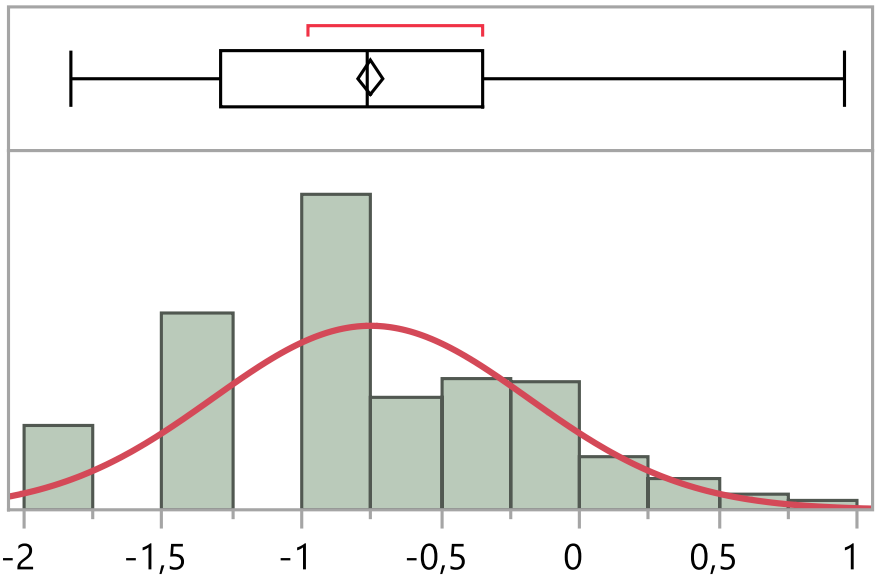


Figure 47 - Johnson Sb distribution [Error to mesh in /m] – Normal (-0,7514,0,56722)

Quantiles

100.0%	maximum	0,9484031092
99.5%		0,766514831
97.5%		0,4468549362
90.0%		-0,006664338
75.0%	quartile	-0,3457484
50.0%	median	-0,764786308
25.0%	quartile	-1,288269874
10.0%		-1,288269874
2.5%		-1,826677018
0.5%		-1,826677018
0.0%	minimum	-1,826677018

Summary Statistics

Mean	-0,751374
Std Dev	0,5672176
Std Err Mean	0,0222481
Upper 95% Mean	-0,707687
Lower 95% Mean	-0,795061
N	650

Fitted Normal
Parameter Estimates

Type	Parameter	Estimate	Lower 95%	Upper 95%
Location	μ	-0,751374	-0,795061	-0,707687
Dispersion	σ	0,5672176	0,5379693	0,5998543

Measure	
-2*LogLikelihood	1106,504
AICc	1110,5226
BIC	1119,458

Goodness-of-Fit Test

Shapiro-Wilk W Test

W	Prob<W
0,971889	<,0001*

- **Distributions FACTOR 1 - CONFIGURATION=Vertical Concave, FACTOR 2 - PROCESSING=Raw**

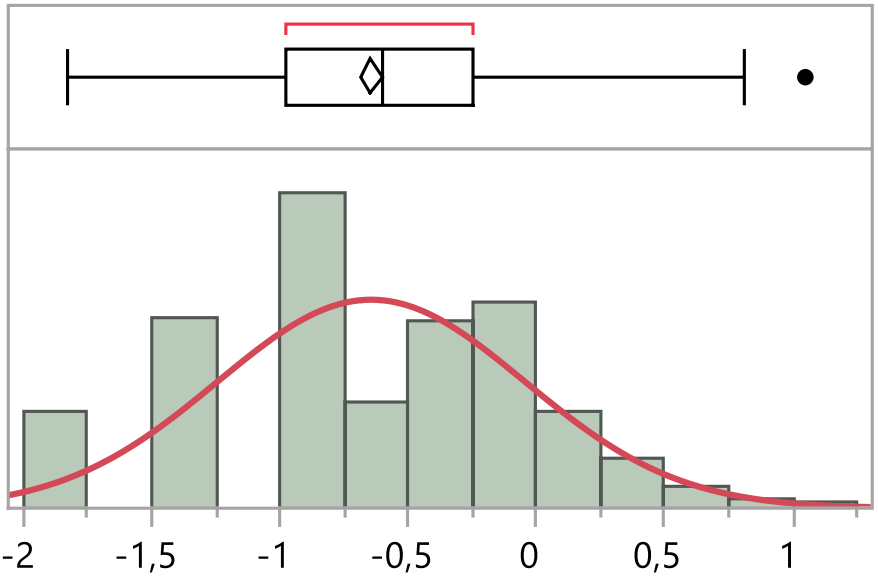


Figure 48 - Johnson Sb distribution [Error to mesh in /m] – Normal (-0,6441,0,59947)

Quantiles

100.0%	maximum	1,0479882044
99.5%		0,7802899974
97.5%		0,4740034703
90.0%		0,1192762696
75.0%	quartile	-0,245740323
50.0%	median	-0,597586225
25.0%	quartile	-0,981033637
10.0%		-1,288269874
2.5%		-1,826677018
0.5%		-1,826677018
0.0%	minimum	-1,826677018

Summary Statistics

Mean	-0,644107
Std Dev	0,599473
Std Err Mean	0,0235133
Upper 95% Mean	-0,597936
Lower 95% Mean	-0,690278
N	650

Fitted Normal
Parameter Estimates

Type	Parameter	Estimate	Lower 95%	Upper 95%
Location	μ	-0,644107	-0,690278	-0,597936
Dispersion	σ	0,599473	0,5685615	0,6339657

Measure	
-2*LogLikelihood	1178,4044
AICc	1182,423
BIC	1191,3584

Goodness-of-Fit Test

Shapiro-Wilk W Test

W	Prob<W
0,976238	<,0001*

4.1.4. Model adjustment

With the transformed data, a new model can be computed. The results can be seen in Figure 49:

Response Johnson Sb[Error_to_mesh/m]
Effect Summary

Source	LogWorth	PValue
FACTOR 1 - CONFIGURATION	666,050	0,00000
FACTOR 2 - PROCESSING	2,384	0,00413
FACTOR 1 - CONFIGURATION*FACTOR 2 - PROCESSING	0,789	0,16254

Summary of Fit

RSquare	0,545736
RSquare Adj	0,545153
Root Mean Square Error	0,67451
Mean of Response	5,29e-16
Observations (or Sum Wgts)	3900

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	2128,3721	425,674	935,6232
Error	3894	1771,6279	0,455	**Prob > F**
C. Total	3899	3900,0000		<,0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	5,288e-16	0,010801	0,00	1,0000
FACTOR 1 - CONFIGURATION[Same Orientation]	1,0207306	0,015275	66,83	<,0001*
FACTOR 1 - CONFIGURATION[Three Passes]	-0,32299	0,015275	-21,15	<,0001*
FACTOR 2 - PROCESSING[Filtered]	-0,030993	0,010801	-2,87	0,0041*
FACTOR 1 - CONFIGURATION[Same Orientation]*FACTOR 2 - PROCESSING[Filtered]	0,0271854	0,015275	1,78	0,0752
FACTOR 1 - CONFIGURATION[Three Passes]*FACTOR 2 - PROCESSING[Filtered]	-0,004545	0,015275	-0,30	0,7660

Effect Tests

Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
FACTOR 1 - CONFIGURATION	2	2	2122,9719	2333,123	<,0001*
FACTOR 2 - PROCESSING	1	1	3,7463	8,2343	0,0041*
FACTOR 1 - CONFIGURATION*FACTOR 2 - PROCESSING	2	2	1,6540	1,8177	0,1625

Figure 49 - Response for transformed Error_to_mesh/m

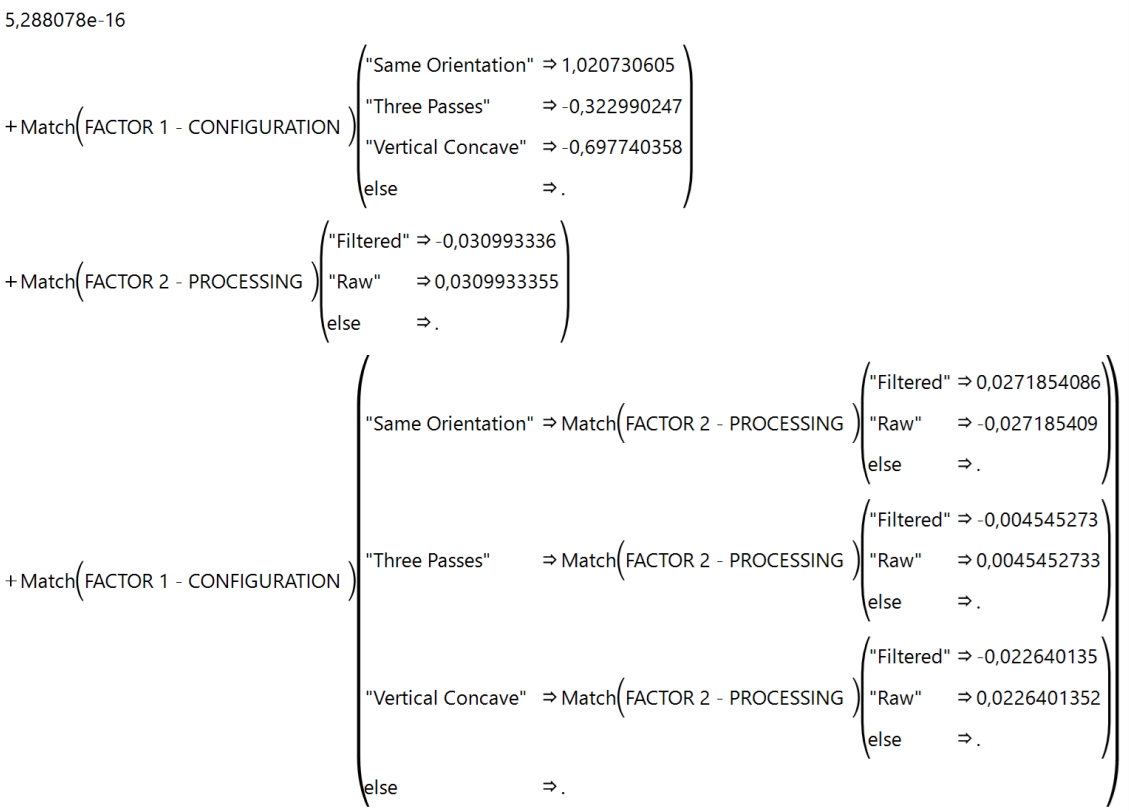


Figure 50 – Prediction expression for transformed Error_to_mesh/m

The results show differences between the different “Processing” and “Configuration” levels, and they do not present any interaction.

4.1.5. Assumption validation (Model 2)

The test still shows that the variance cannot be considered constant, probably due to the large amount of data captured in the point clouds. The transformation has accomplished a better homogeneity regarding the variances, as it can be seen in Figure 51 and Figure 52.

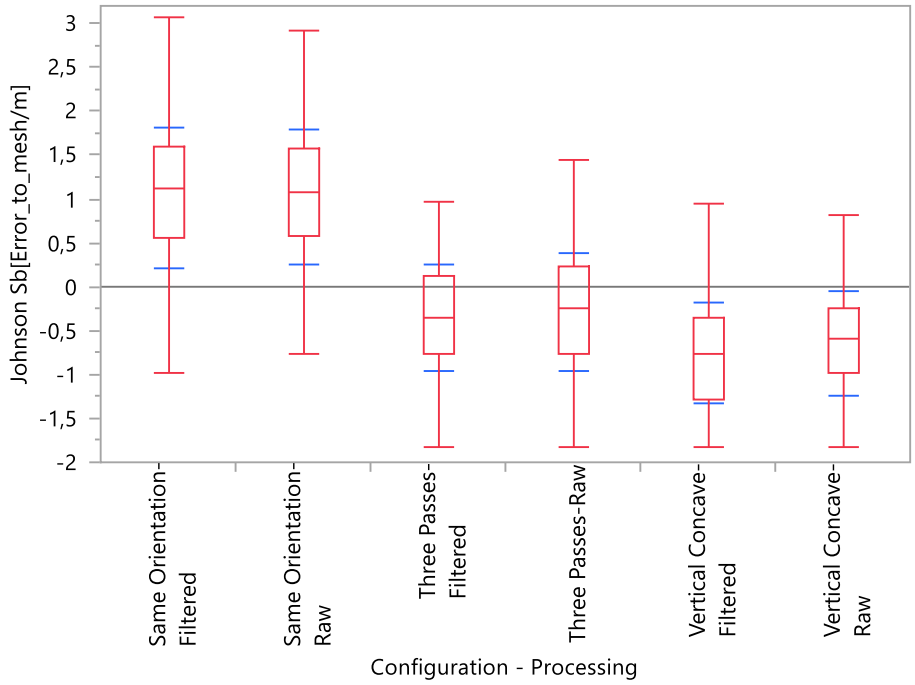


Figure 51 - Oneway analysis of Johnson SB[Error_to_mesh/m] by Configuration - Processing

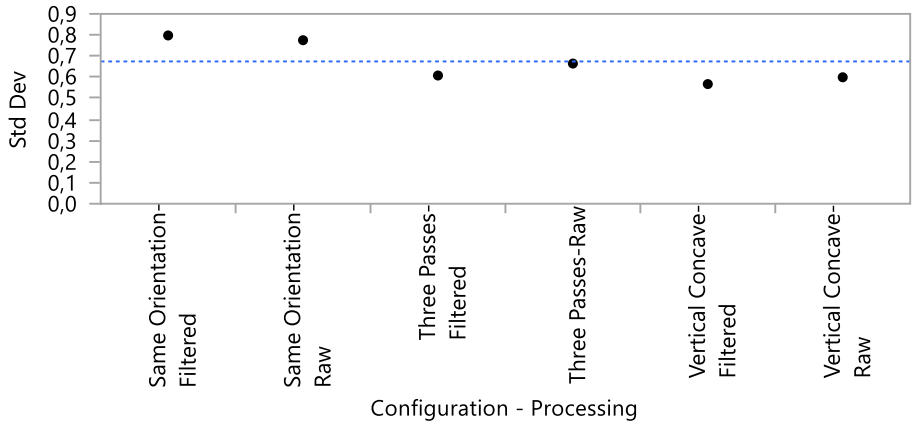


Figure 52 - Test that the variances are equal

Level	Count	Std Dev	MeanAbsDif to Mean	MeanAbsDif to Median
Same Orientation-Filtered	650	0,7977348	0,6250504	0,6193002
Same Orientation-Raw	650	0,7753093	0,6085569	0,6069058
Three Passes-Filtered	650	0,6078934	0,4946173	0,4930443
Three Passes-Raw	650	0,6645733	0,5419206	0,5388404
Vertical Concave-Filtered	650	0,5672176	0,4528337	0,4511829
Vertical Concave-Raw	650	0,5994730	0,4892461	0,4869559

Test	F Ratio	DFNum	DFDen	Prob > F
O'Brien[.5]	24,1666	5	3894	<,0001*
Brown-Forsythe	17,8959	5	3894	<,0001*
Levene	19,0710	5	3894	<,0001*
Bartlett	26,7074	5	.	<,0001*

Welch's Test

Welch Anova testing Means Equal, allowing Std Devs Not Equal

F Ratio	DFNum	DFDen	Prob > F
829,5886	5	1812,9	<,0001*

4.1.6. **Median separation for the transformed data**

- **Regarding the different configurations:**

Means Comparisons

Comparisons for each pair using Student's t

Confidence Quantile

t	Alpha
1,96057	0,05

LSD Threshold Matrix

Abs(Dif)-LSD

	Same Orientation	Three Passes	Vertical Concave
Same Orientation	-0,0519	1,2918	1,6665
Three Passes	1,2918	-0,0519	0,3228
Vertical Concave	1,6665	0,3228	-0,0519

Positive values show pairs of means that are significantly different.

Connecting Letters Report

Level			Mean
Same Orientation	A		1,020731
Three Passes		B	-0,322990
Vertical Concave		C	-0,697740

Levels not connected by same letter are significantly different.

Ordered Differences Report

Level	- Level	Difference	Std Err Dif	Lower CL	Upper CL	p-Value
Same Orientation	Vertical Concave	1,718471	0,026486 5	1,666542	1,770400	<,0001*

Level	- Level	Difference	Std Err Dif	Lower CL	Upper CL	p-Value	
Same Orientation	Three Passes	1,343721	0,026486 5	1,291792	1,395650	<,0001 *	
Three Passes	Vertical Concave	0,374750	0,026486 5	0,322821	0,426679	<,0001 *	

Comparisons for all pairs using Tukey-Kramer HSD
Confidence Quantile

q*	Alpha
2,34460	0,05

HSD Threshold Matrix

Abs(Dif)-HSD

	Same Orientation	Three Passes	Vertical Concave
Same Orientation	-0,0621	1,2816	1,6564
Three Passes	1,2816	-0,0621	0,3126
Vertical Concave	1,6564	0,3126	-0,0621

Positive values show pairs of means that are significantly different.

Connecting Letters Report

Level			Mean
Same Orientation	A		1,020731
Three Passes		B	-0,322990
Vertical Concave		C	-0,697740

Levels not connected by same letter are significantly different.

Ordered Differences Report

Level	- Level	Difference	Std Err Dif	Lower CL	Upper CL	p-Value	
Same Orientation	Vertical Concave	1,718471	0,026486 5	1,65637 1	1,78057 1	<,0001 *	
Same Orientation	Three Passes	1,343721	0,026486 5	1,28162 1	1,40582 1	<,0001 *	
Three Passes	Vertical Concave	0,374750	0,026486 5	0,31265 0	0,43685 0	<,0001 *	

- **Regarding the different processing:**

Means Comparisons
Comparisons for each pair using Student's t
Confidence Quantile

t	Alpha
1,96057	0,05

LSD Threshold Matrix
Abs(Dif)-LSD

	Raw	Filtered
Raw	-0,06277	-0,00079
Filtered	-0,00079	-0,06277

Positive values show pairs of means that are significantly different.
Comparisons for all pairs using Tukey-Kramer HSD
Confidence Quantile

q*	Alpha
1,96057	0,05

HSD Threshold Matrix
Abs(Dif)-HSD

	Raw	Filtered
Raw	-0,06277	-0,00079
Filtered	-0,00079	-0,06277

Positive values show pairs of means that are significantly different.

5. CONCLUSIONS

Photogrammetry is a scanning method that is constantly improving thanks to new algorithms, specially lately thanks to machine learning. This presents it as an appealing method to use in the agricultural field thanks to its demonstrated effectiveness and the relatively low price of the devices used to perform these kinds of scans, when comparing it to other high-end solutions.

The workflow discussed in this final degree project simplifies the process of photogrammetry scanning to a point where there's no need to have a lot of experience in this field to be able to obtain good results.

Regarding the studied variables, the median separation indicates that the best option in order to minimize the error is to use the "Vertical Concave" configuration. The filtered processing also results in significantly better results compared to RAW processing, regarding error compared to the original mesh. These results can be seen in Figure 53 and Figure 54. However, the filtering process can take several hours depending on the number of images used for the photogrammetric reconstruction and the image resolution. The RAW point cloud, on the other

hand, is computed in some minutes. So, if the accuracy of the scan is not critical, or if the scene is clean and easy to compute (which is the case in simulated scans like the ones performed in this project), the point cloud filtering can be omitted.

The plots also corroborate the improved homogeneity mentioned before. “Same Orientation” configuration presents the highest variability, due to the lack of different perspectives to correctly compute the scene depth.

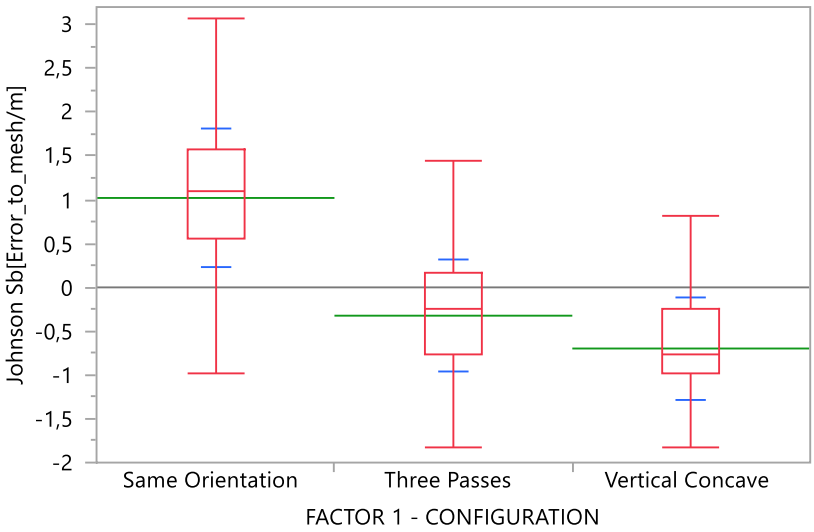


Figure 53 - Box plot for the median separation applied to the "Configuration" variable

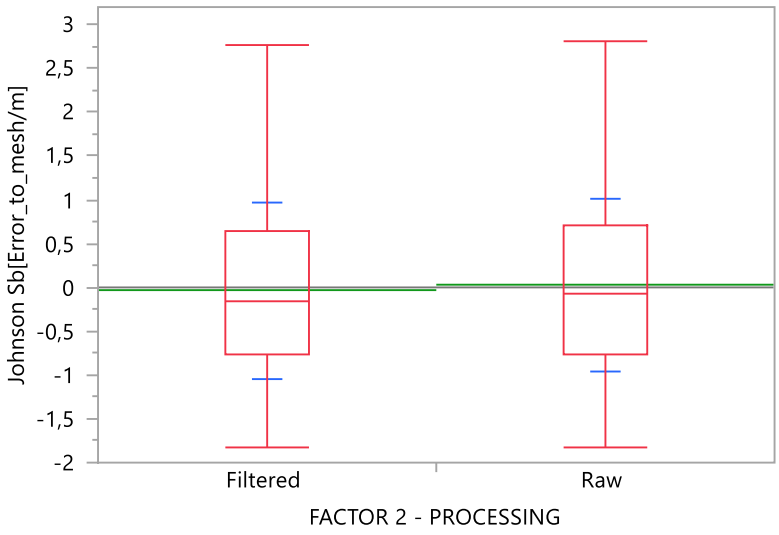


Figure 54 - Box plot for the median separation applied to the "Processing" variable

Regarding the resolution of the images and the density of the point clouds, the conclusion is that the values depend on the specific scenario of each scanning project. If time, hard disk space or processing power can be a restriction, it is better to restrict the resolution of the images to 12 MPx instead of 48 (this is the case for the devices used for the project. Each device will come with its own range of options regarding the integrated camera sensors), and to down-sample the depth maps used to compute the dense point cloud.

6. BIBLIOGRAPHY

AliceVision. (2020, May 20). *AliceVision - Framework Results*. Retrieved from AliceVision - Framework Results: https://alicevision.org/#results

AliceVision. (2020, May 20). *Meshroom - 3D Reconstruction Software*. Retrieved from Meshroom - 3D Reconstruction Software: https://github.com/alicevision/meshroom

AliceVision. (2020, May 20). *Photogrammetry Pipeline - Depth Map Estimation*. Retrieved from Photogrammetry Pipeline - Depth Map Estimation: https://alicevision.org/#photogrammetry/depth_maps_estimation

AliceVision. (2020, May 20). *Photogrammetry Pipeline - Feature Matching*. Retrieved from Photogrammetry Pipeline - Feature Matching: https://alicevision.org/#photogrammetry/feature_matching

AliceVision. (2020, May 20). *Photogrammetry Pipeline - Image Matching*. Retrieved from Photogrammetry Pipeline - Image Matching: https://alicevision.org/#photogrammetry/image_matching

AliceVision. (2020, May 20). *Photogrammetry Pipeline - Meshing*. Retrieved from Photogrammetry Pipeline - Meshing: https://alicevision.org/#photogrammetry/meshing

AliceVision. (2020, May 20). *Photogrammetry Pipeline - Natural Feature Extraction*. Retrieved from Photogrammetry Pipeline - Natural Feature Extraction: https://alicevision.org/#photogrammetry/natural_feature_extraction

AliceVision. (2020, May 20). *Photogrammetry Pipeline - Structure From Motion*. Retrieved from Photogrammetry Pipeline - Structure From Motion: https://alicevision.org/#photogrammetry/sfm

ArcGIS Pro. (2020, May 20). *About feature matching and the match table*. Retrieved from About feature matching and the match table: https://pro.arcgis.com/es/pro-app/tool-reference/editing/about-feature-matching-and-the-match-table.htm

Blender Foundation. (2020, May 20). *Blender Releases*. Retrieved from Blender Releases: https://www.blender.org/download/releases/2-80/

Blender Foundation. (2020, May 20). *Cycles Open Source Production Rendering*. Retrieved from Cycles Open Source Production Rendering: https://www.cycles-renderer.org/

Blender Foundation. (2020, May 20). *Viewport Render*. Retrieved from Viewport Render: https://docs.blender.org/manual/en/latest/editors/3dview/viewport_render.html

Blizard, B. (2020, May 20). *Tested*. Retrieved from Tested: https://www.tested.com/art/makers/460142-art-photogrammetry-how-take-your-photos/

GitHub. (2020, July 3rd). *GitHub Meshroom Discussion*. Retrieved from GitHub Meshroom Discussion: https://github.com/alicevision/meshroom/issues/690

Meshroom. (2020, May 20). *Meshroom Documentation*. Retrieved from Meshroom Documentation: https://meshroom-manual.readthedocs.io/en/latest/index.html#

Saleem, S. A. (2018). A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK.

The Grove 3D. (2020, July 3rd). *The Grove 3D*. Retrieved from The Grove 3D: https://www.thegrove3d.com/

uhlik. (2020, May 20). *Point Cloud Visualizer*. Retrieved from Point Cloud Visualizer: https://github.com/uhlik/bpy

Wikipedia. (2020, May 20). *Alembic (computer graphics)*. Retrieved from Alembic (computer graphics): https://en.wikipedia.org/wiki/Alembic_(computer_graphics)

Wikipedia. (2020, May 20). *Depth map*. Retrieved from Depth map: https://en.wikipedia.org/wiki/Depth_map

Wikipedia. (2020, May 20). *Google Camera*. Retrieved from Google Camera: https://en.wikipedia.org/wiki/Google_Camera

Wikipedia. (2020, May 20). *High-dynamic-range imaging*. Retrieved from High-dynamic-range imaging: https://en.wikipedia.org/wiki/High-dynamic-range_imaging

Wikipedia. (2020, July 3rd). *Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Structure_from_motion

7. ANNEXES

7.1. Image Feature Extraction algorithms – Detailed explanation

All the information in this section is extracted/based from (Saleem, 2018).

- **SIFT**

The most renowned feature-detection-description algorithm is the Scale Invariant Feature Transform (SIFT), which is based on a Laplacian-of-Gaussian (LoG) approximated with a Difference-of-Gaussians (DoG) operator. By using DoG at various scales of the subject images it can detect feature points by searching local maxima. The description method then extracts a 16x16 neighborhood around each detected feature and further segments the region into sub-blocks, rendering a total of 128 bin values.

Equation (4) shows the convolution of difference of two Gaussians (computed at different scales) with image "I (x, y)".

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) \cdot I(x, y) \tag{4}$$

Where: $G \rightarrow$ *Gaussian function*

- **SURF**

Speeded Up Robust Features (SURF) is an algorithm that relies on Gaussian scale space analysis of images, based on Hessian Matrix determinant exploiting integral images in order to improve the speed of the feature-detection.

Equation (5) represents the Hessian Matrix in point "x = (x,y)" for a set scale "σ".

$$\mathcal{H}(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{yx}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{5}$$

Where: $L_{xx}(x, \sigma) \rightarrow$ *Convolution of the second order derivative of the Gaussian with the image* $\mathbb{I}$ *in point* x

- **KAZE**

The features associated to this algorithm take advantage of the non-linear scale space through non-linear diffusion filtering, making blurring in images locally adaptive to feature-points. This leads to noise reduction while maintaining the subject image regions' boundaries.

The KAZE feature detector is based on a *Hessian Matrix scale normalized determinant* computed at several scale levels, picking up feature-points using a moving window with the maxima of detector response. By finding the dominant orientation in a circular neighborhood around each detected feature the algorithm achieves rotation invariance.

Equation (6) shows the formula for the standard nonlinear diffusion.

$$\frac{\partial L}{\partial t} = div(c(x, y, t).\nabla L)$$

$$Where: c \rightarrow Conductivity\ function$$

$$div \rightarrow Divergence \tag{6}$$

$$\nabla \rightarrow Gradient\ operator$$

$$L \rightarrow Image\ luminance$$

- **AKAZE**

This algorithm is based on the same principle as KAZE, but it uses a computationally efficient framework to construct its non-linear scale spaces. The framework is called *Fast Explicit Diffusion (FED)* and the AKAZE feature detector is based on a *Hessian Matrix determinant*.

AKAZE uses *Scharr filters* to improve rotation invariance quality. Feature-points are picked using maxima of the detector responses in spatial location. The highly efficient *Modified Local Difference Binary (MLDB)* algorithm is used as a base for the AKAZE descriptor.

- **ORB**

Modified FAST (Features from Accelerated Segment Test) detection and direction-normalized BRIEF (Binary Robust Independent Elementary Features) description methods are blended to obtain the Oriented FAST and Rotated BRIEF (ORB) algorithm.

A modified version of BRIEF descriptor is used since the standard BRIEF description method is highly unstable with rotation changes.

- **BRISK**

AGAST algorithm and *FAST Corner score* filtering is used in the *Binary Robust Invariant Scalable Keypoints (BRISK) algorithm to* detect corners. BRISK description identifies the characteristic direction of each feature to achieve rotation invariance, while simple brightness tests are also concatenated to cater illumination invariance.

7.1.1. Algorithm comparison – Detailed explanation

In order to compare the different algorithms, some image pairs are used. Both images are taken from a different perspective and/or rotation, and the algorithms have to join them correctly, as seen with SIFT in Figure 55.

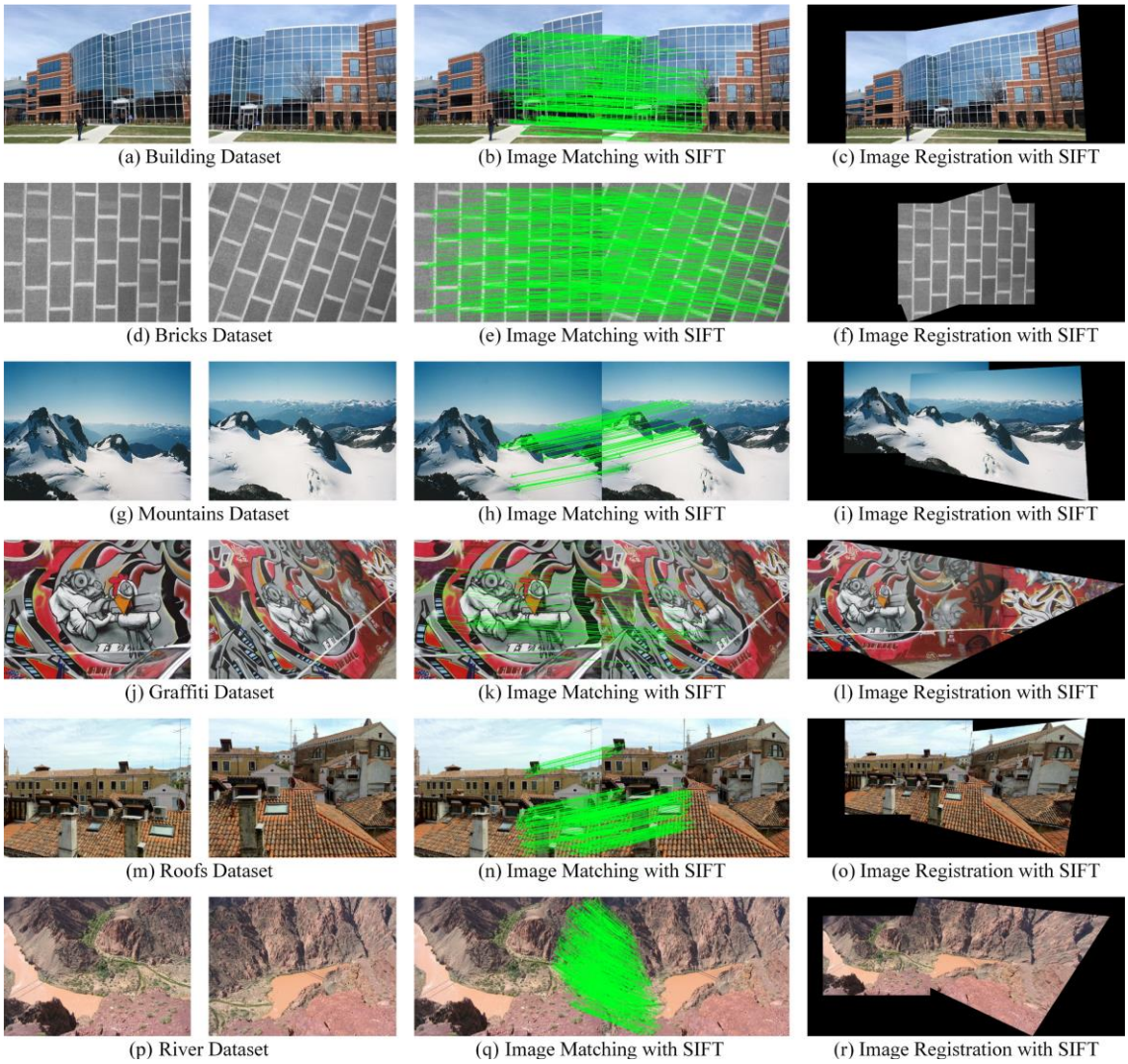


Figure 55. Image pairs selected from different benchmark datasets. Image registration and mosaicking has been performed using SIFT algorithm. (Saleem, 2018)

In Figure 56, an example of algorithm comparison is presented, to better illustrate the procedure and results.

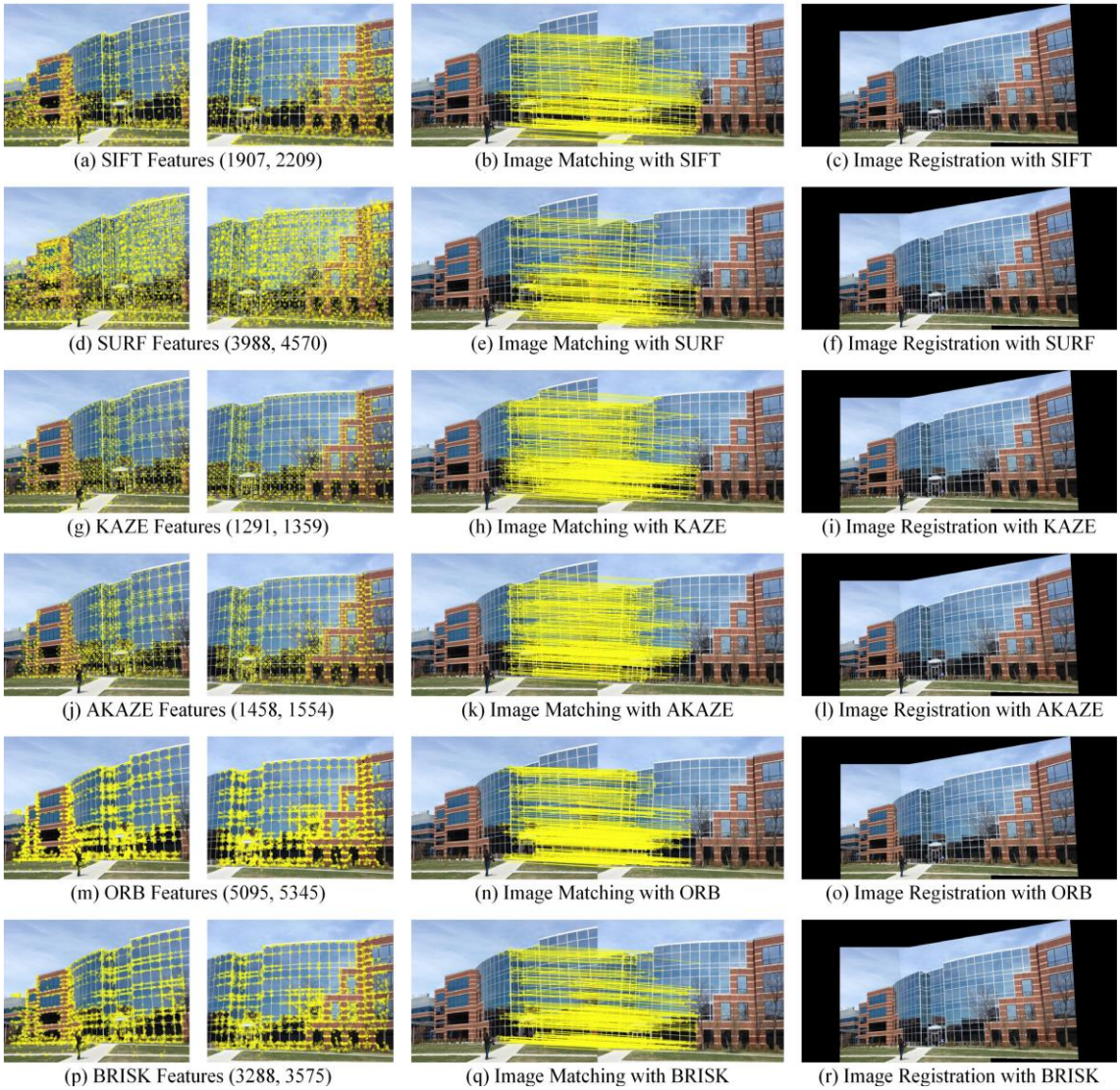


Figure 56. Feature-detection, matching, and mosaicing with SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. (Saleem, 2018)

- **Quantitative comparison**
    - More features are paired by SURF (64D) in comparison to SURF (128D).
    - The SIFT and SURF features are detected in a scattered manner, usually throughout the image, while ORB and BRISK features are more concentrated in the corners.
    - ORB detects the largest number of features.
    - BRISK is second place because it detects a greater number of features than SIFT, AKAZE, SURF and KAZE.
    - The least number of features were detected by KAZE.
    - SURF detects more features than SIFT.
    - We get more detected features with AKAZE over KAZE.
- **Feature-Detection-Description Time comparison**
    - ORB has the highest efficiency among all the feature descriptor-detector algorithms, having the lowest computational cost.
    - The BRISK algorithm also has computational efficiency, but it's a bit more expensive than ORB.
    - SURF (128D) and SURF (64D) have increased efficiency over SIFT (128D).
    - AKAZE achieves better computational efficiency than SIFT, KAZE, SURF (128D) and SURF (64D), but less than BRISK and ORB.
    - Feature-detection-description for KAZE has the highest computational cost.
    - SIFT computational cost is less than 50% the computational cost of KAZE.
- **Feature Matching Time**
    - SURF (128D) has the highest computational cost for feature matching.
    - SIFT is the second most computationally demandant algorithm.
    - Compared to SIFT, SURF (64D) has a lower computational cost for feature matching.
    - ORB and BRISK detect many features in unlimited state, but their feature matching cost increases drastically. Using these detectors (so, ORB (1000) and BRISK (1000)) in a bounded state, this cost can be reduced.
    - ORB (1000) has the lowest feature matching cost.
    - Feature descriptors *matching* computational cost for KAZE is lower than SIFT, SURF (128D), SURF (64D), ORB and BRISK.
    - Features matching cost for AKAZE is less than KAZE.

Algorithm	Features Detected in the Image Pairs		Features Matched	Outliers Rejected	Feature Detection & Description Time (s)		Feature Matching Time (s)	Outlier Rejection & Homography Calculation Time (s)	Total Image Matching Time (s)
	1st Image	2nd Image			1st Image	2nd Image			
Building Dataset (Image Pair # 1)									
SIFT	1907	2209	384	51	0.1812	0.1980	0.1337	0.0057	0.5186
SURF(128D)	3988	4570	319	58	0.1657	0.1786	0.5439	0.0058	0.8940
SURF(64D)	3988	4570	612	73	0.1625	0.1734	0.2956	0.0052	0.6367
KAZE	1291	1359	465	26	0.2145	0.2113	0.0613	0.0053	0.4924
AKAZE	1458	1554	475	36	0.0695	0.0715	0.0307	0.0055	0.1772
ORB	5095	5345	854	149	0.0213	0.0220	0.1586	0.0067	0.2086
ORB(1000)	1000	1000	237	21	0.0103	0.0101	0.0138	0.0049	0.0391
BRISK	3288	3575	481	47	0.0533	0.0565	0.1236	0.0056	0.2390
BRISK(1000)	1000	1000	190	33	0.0188	0.0191	0.0158	0.0049	0.0586
Bricks Dataset (Image Pair # 2)									
SIFT	1404	1405	427	16	0.1571	0.1585	0.0680	0.0053	0.3889
SURF(128D)	2855	2332	140	34	0.1337	0.1191	0.2066	0.0051	0.4645
SURF(64D)	2855	2332	327	63	0.1307	0.1137	0.1173	0.0055	0.3672
KAZE	366	705	88	6	0.1930	0.1988	0.0105	0.0047	0.4070
AKAZE	278	289	153	9	0.0541	0.0536	0.0037	0.0049	0.1163
ORB	978	942	323	17	0.0075	0.0079	0.0124	0.0049	0.0327
ORB(1000)	731	734	240	32	0.0067	0.0073	0.0088	0.0050	0.0278
BRISK	796	752	285	18	0.0146	0.0144	0.0119	0.0049	0.0458
BRISK(1000)	796	752	285	18	0.0146	0.0144	0.0119	0.0049	0.0458
Mountain Dataset (Image Pair # 3)									
SIFT	1867	2033	170	45	0.1943	0.2017	0.1197	0.0047	0.5204
SURF(128D)	1890	2006	175	33	0.0979	0.1130	0.1208	0.0051	0.3368
SURF(64D)	1890	2006	227	62	0.0978	0.1113	0.0689	0.0053	0.2833
KAZE	972	971	131	20	0.2787	0.2826	0.0343	0.0050	0.6006
AKAZE	960	986	187	16	0.0841	0.0842	0.0151	0.0050	0.1884
ORB	4791	5006	340	78	0.0228	0.0236	0.1397	0.0052	0.1913
ORB(1000)	1000	1000	113	29	0.0118	0.0118	0.0117	0.0049	0.0402
BRISK	3153	3382	287	22	0.0520	0.0555	0.1083	0.0052	0.2210
BRISK(1000)	1000	1000	143	7	0.0201	0.0213	0.0160	0.0046	0.0620
Graffiti Dataset (Image Pair # 4)									
SIFT	2654	3698	99	51	0.2858	0.3382	0.2940	0.0073	0.9253
SURF(128D)	4802	5259	44	31	0.2166	0.2184	0.7399	0.0222	1.1971
SURF(64D)	4802	5259	62	42	0.2072	0.2142	0.3951	0.0169	0.8334
KAZE	2232	2302	30	9	0.3311	0.3337	0.1594	0.0052	0.8294
AKAZE	2064	2205	23	13	0.1158	0.1185	0.0491	0.0081	0.2915
ORB	5527	7517	38	22	0.0277	0.0341	0.2129	0.0083	0.2830
ORB(1000)	1000	1000	16	7	0.0146	0.0157	0.0114	0.0059	0.0476
BRISK	3507	5191	54	22	0.0669	0.0953	0.1687	0.0077	0.3386
BRISK(1000)	1000	1000	18	10	0.0227	0.0239	0.0143	0.0073	0.0682
Roofs Dataset (Image Pair # 5)									
SIFT	2303	3550	423	154	0.1983	0.2665	0.2475	0.0063	0.7186
SURF(128D)	2938	3830	171	95	0.1173	0.1523	0.3349	0.0084	0.6129
SURF(64D)	2938	3830	247	143	0.1165	0.1504	0.1847	0.0090	0.4606
KAZE	1260	1736	172	85	0.2119	0.2265	0.0710	0.0068	0.5162
AKAZE	1287	1987	175	59	0.0686	0.0806	0.0294	0.0053	0.1839
ORB	7660	11040	498	157	0.0296	0.0407	0.4131	0.0065	0.4899
ORB(1000)	1000	1000	91	32	0.0106	0.0113	0.0111	0.0055	0.0385
BRISK	5323	7683	436	207	0.0899	0.1260	0.3672	0.0074	0.5905
BRISK(1000)	1000	1000	90	44	0.0189	0.0199	0.0156	0.0069	0.0613
River Dataset (Image Pair # 6)									
SIFT	8619	9082	1322	192	0.6795	0.7083	2.2582	0.0092	3.6552
SURF(128D)	9434	10471	223	63	0.3768	0.4205	2.8521	0.0055	3.6549
SURF(64D)	9434	10471	386	66	0.3686	0.4091	1.4905	0.0049	2.2731
KAZE	2984	2891	391	78	0.5119	0.5115	0.2669	0.0059	1.2962
AKAZE	3751	3635	376	65	0.2048	0.1991	0.1341	0.0056	0.5436
ORB	34645	35118	2400	553	0.1219	0.1276	5.3813	0.0092	5.6400
ORB(1000)	1000	1000	40	6	0.0235	0.0235	0.0109	0.0046	0.0625
BRISK	23607	24278	1725	366	0.3813	0.4040	4.8117	0.0089	5.6059
BRISK(1000)	1000	1000	39	8	0.0251	0.0248	0.0155	0.0040	0.0694
Mean Values for All Image Pairs									
SIFT	3125.7	3662.8	470.8	84.8	0.2827	0.3119	0.5202	0.0064	**1.1212**
SURF(128D)	4317.8	4744.7	178.7	52.3	0.1847	0.2003	0.7997	0.0087	**1.1934**
SURF(64D)	4317.8	4744.7	310.2	74.8	0.1806	0.1954	0.4254	0.0078	**0.8092**
KAZE	1517.5	1660.7	212.8	37.3	0.2902	0.2941	0.1006	0.0055	**0.6904**
AKAZE	1633.0	1776.0	231.5	33.0	0.0995	0.1013	0.0437	0.0057	**0.2502**
ORB	9782.7	10828.0	742.2	162.7	0.0385	0.0427	1.0530	0.0068	**1.1410**
ORB(1000)	955.2	955.7	122.8	21.2	0.0129	0.0133	0.0113	0.0051	**0.0426**
BRISK	6612.3	7476.8	544.7	113.7	0.1097	0.1253	0.9319	0.0066	**1.1735**
BRISK(1000)	966.0	958.7	127.5	20.0	0.0200	0.0206	0.0149	0.0054	**0.0609**

Table 13. Quantitative comparison and computational costs of different feature-detector-descriptors/algorithms (Saleem, 2018)

- **Outlier Rejection and Homography Fitting Time**

Outliner-rejection and homography fitting time is approximately the same for all feature-detector-descriptors.

- **Total Image Matching Time**
    - The fastest image matching time is provided by BRISK (1000) and ORB (1000).
    - Due to the lower feature-matching cost of KAZE, the overall image matching time taken by this algorithm is less than SURF (64D), SURF (128D) and SIFT.
    - SURF (128D) algorithm takes longer for the overall image matching process over SIFT (128D).
    - SIFT (128D) takes longer than SURF (64D) for the overall image matching.
    - ORB and BRISK (in unbounded state) have a considerably low computational cost for feature-detection-description but matching this large quantity of features increases their overall image matching time.

Algorithm	Mean Feature-Detection-Description Time per Point (µs)		Mean Feature Matching Time per Point (µs)
	1st Images	2nd Images	
SIFT	90.44	85.15	**142.02**
SURF(128D)	42.78	42.22	**168.55**
SURF(64D)	41.83	41.18	89.66
KAZE	**191.24**	**177.09**	60.58
AKAZE	60.93	57.04	24.61
ORB	**3.94**	**3.94**	97.25
ORB(1000)	13.51	13.92	**11.82**
BRISK	16.59	16.76	**124.64**
BRISK(1000)	20.70	21.49	**15.42**

Table 14. Computational cost per feature point based on mean values for all image pairs (Saleem, 2018)

- **Repeatability**
    - SIFT, SURF, KAZE, AKAZE, and BRISK repeatability remains high for up scaling of images. For down scaling, KAZE, AKAZE, and BRISK repeatability drops significantly.
    - Repeatability of SIFT and SURF remains steady for scale variations, while SURF's drops at around **10%** scale. Repeatability has remained stable for SIFT when down scaling to as low as **3%.** Meanwhile, the repeatability of the other algorithms gets near zero for scales of **15%** and below.
    - ORB (1000) and BRISK (1000) have higher repeatability than ORB and BRISK for image rotations and up scaling while the case is reversed for down scaling.
    - If the scale of the *test image* remains in the range of **60% to 150%** with respect to the *reference image,* image matching repeatability for ORB algorithm remains stable and high. Beyond this range, its repeatability drops considerably.
    - ORB (1000), BRISK (1000), AKAZE, ORB, and KAZE have a higher repeatability than SIFT and SURF for image rotations.
    - Repeatability of ORB and BRISK are generally higher than others for affine changes.

- **Accuracy of Image Matching**
    - SIFT is the most accurate algorithm for rotation, scale and affine variations.
    - BRISK is the second-best algorithm regarding accuracy of image matching, with respect to the accuracy for rotation and scale changes.
    - For image rotations and scale variations, AKAZE's accuracy is comparable to BRISK while in the range of **40% to 400%**. Its accuracy decreases beyond this range.
    - ORB (1000) is less accurate for scale and rotation changes than BRISK (1000).
    - For affine changes, ORB (1000) and BRISK (1000) are comparable.
    - ORB and BRISK are more accurate than ORB (1000) and BRISK (1000).
    - For scale and affine variations, AKAZE is more accurate than KAZE.
    - KAZE is more accurate than AKAZE for rotation changes.

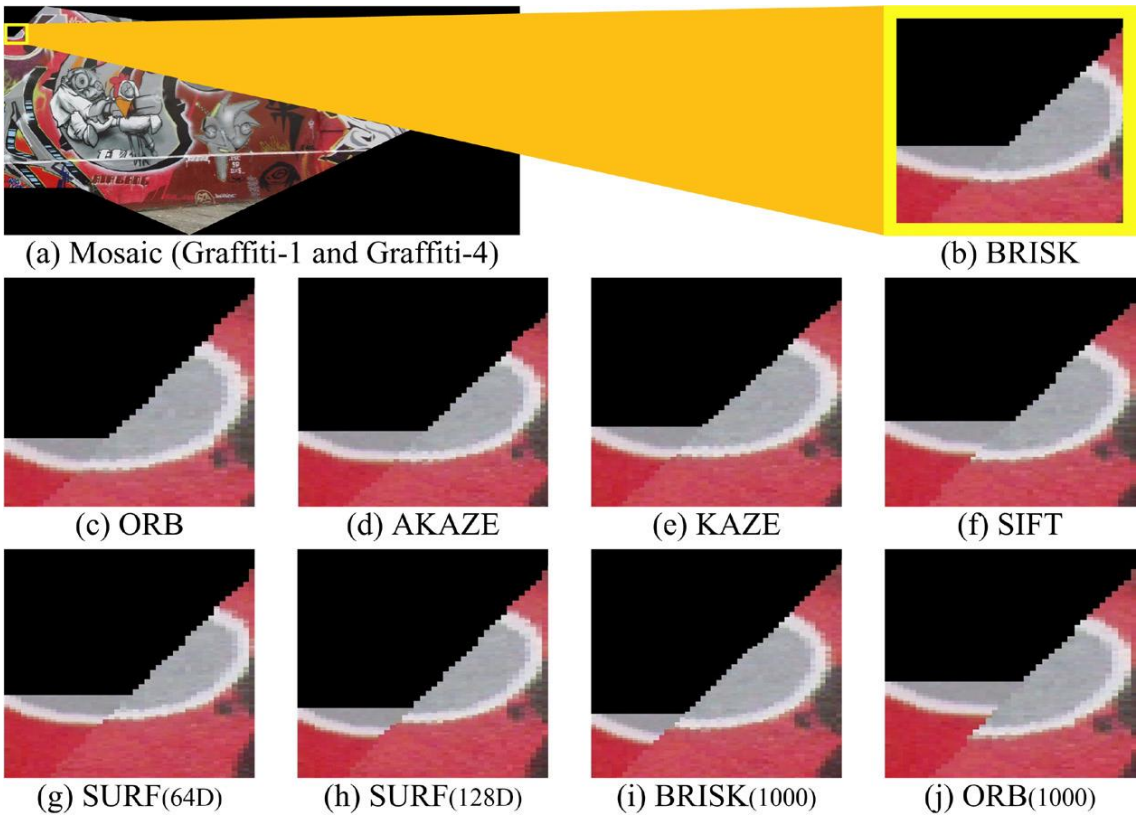


Figure 57. Illustration of image registration error using the feature-detector-descriptors for Graffiti image pair (1,4). Based on observation, BRISK provides best accuracy particularly for this image set. Notice that the accuracy of ORB (1000) and BRISK (1000) is less than ORB and BRISK, respectively. (Saleem, 2018)

- **Result analysis**

Based on repeatability, the most scale invariant feature detectors are found to be SIFT, SURF, and BRISK, as they have survived widespread scale variations, while ORB is the least scale invariant algorithm. Meanwhile, the most rotation invariant feature detectors are found to be ORB (1000), BRISK (1000), and AKAZE. On the other hand, the most affine change invariant algorithms are found to be ORB and BRISK as compared to others.

Compared to the rest for image rotations, SIFT, KAZE, AKAZE, and BRISK have a higher accuracy. Although the most efficient algorithms are ORB and BRISK, since they can detect a huge number of features, the matching time for so many features make their *total image matching time* longer. ORB (1000) and BRISK (1000), on the contrary, perform the fastest matching of the images, but they get compromised results for accuracy. SIFT and BRISK is found to have the highest overall accuracy for all types of geometric transformations

As a conclusion, SIFT is the most accurate algorithm.

7.2. Full GitHub discussion regarding .ply conversion

This appendix is a compilation of all the messages shared in the GitHub discussion regarding .ply conversion in Meshroom. (GitHub, 2020)

Edit | New issue | Jump to bottom

Add option to export dense point cloud to other formats (Workaround here) #690

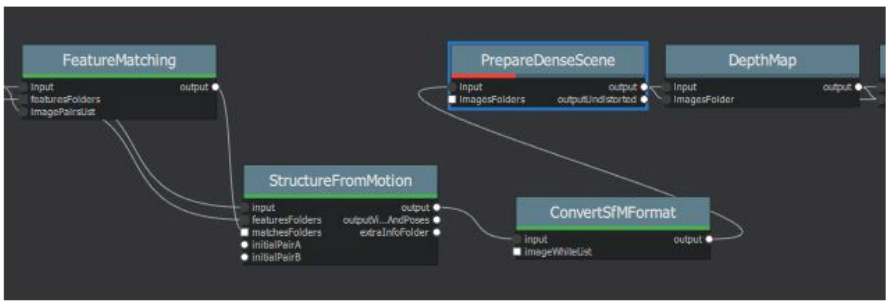
Closed **miquelrosell99** opened this issue on 2 Nov 2019 · 54 comments

Labels: type:question

miquelrosell99 commented on 2 Nov 2019

Describe the bug

PrepareDenseScene node fails for ConvertSfMFormat input in .ply, so i can only export .ply of low density point clouds



miquelrosell99 added the bug label on 2 Nov 2019

natowi commented on 2 Nov 2019 — Member

You can simply change the Inter File Extension from abc to ply in the StructureFromMotion node (Advanced settings enabled) so you don´t need the ConvertSfMFormat node in this case.

1 | 1

miquelrosell99 commented on 2 Nov 2019 • edited — Author

@miquelrosell99 unknown Describer Type should allready be available in your downloaded dev version of MR

+😀

miquelrosell99 commented on 16 Nov 2019 Author

> **@miquelrosell99** Good that the workaround works for you. I closed the issue, since ConvertSfM should work, and modifying all the parameters is only a workaround and ply is not supported by the Texturing node and 3D viewer.
>
> We can use ConvertSfM but need to disable SIFT and enable unknown Describer Types
> *Note: not available in the binaries, option recently added

Any place I can get info on describer types?

+😀

miquelrosell99 commented on 16 Nov 2019 Author

> **@miquelrosell99** unknown Describer Type should allready be available in your downloaded dev version of MR

Yes it is. But I'll download latest official one, since the only change I added was to get ply files, and i can do that with official. Better for future updates

+😀

natowi commented on 16 Nov 2019 • edited Member

Any place I can get info on describer types?

They are feature detection algorithms. SIFT for example: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
https://www.researchgate.net/publication/323561586_A_comparative_analysis_of_SIFT_SURF_KAZE_AKAZE_ORB_and_BRISK

+😀

natowi removed the scope:doc label on 28 Jan

natowi commented on 28 Jan Member

At the moment there is the problem left, that

```
class Meshing(desc.CommandLineNode):
    commandLine = 'aliceVision_meshing {allParams}'
```

calls all Params name=(...) but my file type selector is not a valid param

```
desc.ChoiceParam(
            name='outputDensePointCloud',
...
```

I think this can be fixed using all supported params, leaving out outputDensePointCloud:

```
class Meshing(desc.CommandLineNode):
    commandLine = 'aliceVision_meshing --input {inputValue} ...
```

I did not test this yet.

+😀

natowi commented on 16 Nov 2019 • edited ▾ Member

@miquelrosell99 Good that the workaround works for you. I closed the issue, since ConvertSfM should work, and modifying all the parameters is only a workaround and ply is not supported by the Texturing node and 3D viewer.

We can use ConvertSfM but need to disable SIFT and enable unknown Describer Types
*Note: not available in the binaries, option recently added

+😀

natowi added scope:doc type:question and removed feature request labels on 16 Nov 2019

miquelrosell99 commented on 16 Nov 2019 Author

Nice! Gonna download latest files then

+😀

natowi commented on 16 Nov 2019 Member

+😀

miquelrosell99 commented on 10 Nov 2019 • edited ▾ Author

Yep, problem is it only converts camera locations
Processing meshing node rn, but its been stuck here for about 2h, with 100% ram usage

```
638 [16:08:26.525426][info]           - size[5]: 14
639 [16:08:26.525426][info]           - size[7]: 108
640 [16:08:26.525426][info]           - size[9]: 26285206
641 [16:08:26.525426][info]           - size[112653]: 1
642 [16:08:26.525426][info]           - size[26172675]: 1
643 [16:08:26.525426][info]           - capacity[9]: 26285328
644 [16:08:26.525926][info]           - capacity[26285328]: 2
645 [16:08:26.525926][info] Compute boykov_kolmogorov_max_flow.
```

With depth map downsample to 2 it works great

+😀

miquelrosell99 commented on 10 Nov 2019 • edited ▾ Author

@natowi gonna push the commit if you want to use it ;) Will be in my fork of Meshroom
Edit: https://github.com/miquelrosell99/meshroom/commit/3479de9920ad0f6aec555883c93d8ddb385ab3b8

+😀

miquelrosell99 commented on 16 Nov 2019 • edited ▾ Author

@natowi **@fabiencastan** Hi, how can I add these changes to official meshroom repo? Now I have them on my own fork.

Commits on Nov 10, 2019

miquelrosell99	Add Meshroom GUI launcher	42d8827
natowi and miqu...	Add file type selection for dense point cloud ...	46d55ce
natowi and miqu...	Fixed file type	acdad4d
natowi and miqu...	Use outputSfMDataFilename	b083839
natowi and miqu...	Use desc.Node.internalFolder	bf36d64
natowi and miqu...	Update Meshing.py	1557111
natowi and miqu...	Fix file extension	524d6a0
miquelrosell99	Add support for selecting custom file type for dense point cloud	3479de9

I fixed what was discussed here and it works great

> I have changed the code since (check the PRfor details).

+😀

natowi commented on 10 Nov 2019 • edited Member

You added only the value variables {???Value}. They also need the --command prefix as listed above.
--??? {???Value}

1 +😀

miquelrosell99 commented on 10 Nov 2019 Author

Okay now it works. I'll put all the proper names to optional parameters

+😀

fabiencastan commented on 10 Nov 2019 • edited Member

Do not modify the full pipeline to use another file format, just use ConvertSfMFormat node where you need to convert your Alembic file to something else.
Most of the file formats are not able to store all the information we need: cameras, visibilities, etc.
For instance, PLY is only able to deal with point clouds without extra-attributes.

+😀

natowi commented on 10 Nov 2019 • edited Member

@fabiencastan

miquelrosell99

> Meshing node only outputs .abc file, and I can't manage to use ConvertSfMFormat node there. It only exports the camera dots inside the .ply. No actual object points

(I assumed ConvertSfMFormat would only work with StructureFromMotion since the node was not called ConvertPointCloudFormat ;))

StructureFromMotion sfm.abc -> ConvertSfMFormat will convert the whole pointcloud

Meshing densePointCloud.abc -> ConvertSfMFormat will only convert the cameras and dismiss the pointcloud

When comparing .json conversions, the whole "structure": [ ] class is missing when converting from a meshing .abc file. Maybe there is something wrong with the meshing node .abc files?

Advanced parameters:
--universePercentile arg (=0.999) universe percentile
--estimateSpaceMinObservations arg (=3)
Minimum number of observations for SfM
space estimation.
--estimateSpaceMinObservationAngle arg (=10)
Minimum angle between two observations
for SfM space estimation.
--pixSizeMarginInitCoef arg (=2) pixSizeMarginInitCoef
--pixSizeMarginFinalCoef arg (=1) pixSizeMarginFinalCoef
--voteMarginFactor arg (=4) voteMarginFactor
--contributeMarginFactor arg (=2) contributeMarginFactor
--simGaussianSizeInit arg (=10) simGaussianSizeInit
--simGaussianSize arg (=10) simGaussianSize
--minAngleThreshold arg (=0.10000000000000001)
minAngleThreshold
--refineFuse arg (=1) refineFuse
--saveRawDensePointCloud arg (=0) Save dense point cloud before cut and
filtering.

Log parameters:
-v [--verboseLevel] arg (=info) verbosity level (fatal, error, warning,
info, debug, trace).

+😀

natowi commented on 10 Nov 2019 — Member

There are a lot parameters, this is why I have not tested this yet.

+😀

miquelrosell99 commented on 10 Nov 2019 — Author

But I already have all these parameters added. outputMesh is there in the code

+😀

miquelrosell99 commented on 10 Nov 2019 — Author

Tag me when you test it please

+😀

miquelrosell99 commented on 10 Nov 2019 Author

And for Meshing?

+😀

natowi commented on 10 Nov 2019 Member

Required parameters:
-i [--input] arg SfMData file.
-o [--output] arg Output Dense SfMData file.
-o [--outputMesh] arg Output mesh (OBJ file format).

Optional parameters:
--depthMapsFolder arg Input depth maps folder.
--depthMapsFilterFolder arg Input filtered depth maps folder.
--maxInputPoints arg (=50000000) Max input points loaded from images.
--maxPoints arg (=5000000) Max points at the end of the depth maps
fusion.
--maxPointsPerVoxel arg (=6000000) Max points per voxel.
--minStep arg (=2) The step used to load depth values from
depth maps is computed from
maxInputPts. Here we define the minimal
value for this step, so on small
datasets we will not spend too much
time at the beginning loading all depth
values.
--simFactor arg (=15) simFactor
--angleFactor arg (=15) angleFactor
--partitioning arg (=1) Partitioning: 'singleBlock' or 'auto'.
--repartition arg (=1) Repartition: 'multiResolution' or
'regularGrid'.
--estimateSpaceFromSfM arg (=1) Estimate the 3d space from the SfM.
--addLandmarksToTheDensePointCloud arg (=0)
Add SfM Landmarks into the dense point
cloud (created from depth maps). If
only the SfM is provided in input, SfM
landmarks will be used regardless of
this option.
--colorizeOutput arg (=0) Whether to colorize output dense point
cloud and mesh.

After adding this :
commandLine = 'aliceVision_meshing --input {inputValue} {depthMapsFolderValue} {depthMapsFilterFolderValue} {estimateSpaceFromSfMValue} {estimateSpaceMinObservationsValue} {estimateSpaceMinObservationAngleValue} {maxInputPointsValue} {maxPointsPerVoxelValue} {minStepValue} {partitioningValue} {repartitionValue} {angleFactorValue} {simFactorValue} {pixSizeMarginInitCoefValue} {pixSizeMarginFinalCoefValue} {voteMarginFactorValue} {contributeMarginFactorValue} {simGaussianSizeInitValue} {simGaussianSizeValue} {minAngleThresholdValue} {refineFuseValue} {addLandmarksToTheDensePointCloudValue} {colorizeOutputValue} {saveRawDensePointCloudValue} {verboseLevelValue} --output {outputValue} {outputMeshValue}'
I get this error @natowi

```
ERROR:root:Error during Graph execution Error on node "Meshing_1":
Log:
ERROR: the option '--outputMesh' is required but missing
```

+

natowi commented on 10 Nov 2019 — Member

{depthMapsFolderValue} will only add the /value/, so you need to look up all the --params and add them like this --input {inputValue}

+

miquelrosell99 commented on 10 Nov 2019 • edited — Author

I dont understand... HEre you use it like this

```
class InstantMeshes(desc.CommandLineNode):
    commandLine = 'alicevision_InstantMeshes {inputValue} -S {smoothValue} --output {outputValue}'

    cpu = desc.Level.NORMAL
    ram = desc.Level.NORMAL
```

In my last comment you have all the parameters in the node. Could you tell me what to add ro remove? I dont understand how to program Meshroom. I've only used intuition

+

natowi commented on 10 Nov 2019 • edited — Member

Yes, for Instant Meshes the expected CLI command is

```
<inputpath> -S <smoothvalue> --output <outputpath>
```

PR=Pull request

My last update was `value=desc.Node.internalFolder + 'densepointcloud.{outputDenseFileTypeValue}',`

`commandLine = 'aliceVision_meshing --input {inputValue}` was my example.

`commandLine = 'aliceVision_meshing` starts the CLI and aliceVision_meshing.exe with the following parameters (by default {allParams}). allParams will look for all `desc.???Param(` and adds the name and value this way to the cli: `aliceVision_meshing.exe --input c:\sfm.abc ...`

By adding

```
desc.ChoiceParam(
            name='outputDenseFileType',
```

and leaving {allParams} `--outputDenseFileType value` is added to the cli, but the command does not exist. I think this can be fixed by listing only the supported commands instead of {allParams}.

Here is a working example for a new node:
https://github.com/natowi/meshroom_external_plugins/blob/master/InstantMeshes.py

+😀

miquelrosell99 commented on 10 Nov 2019 Author

I see. So we would need to add all parameters manually?

+😀

miquelrosell99 commented on 10 Nov 2019 Author

@natowi This error appears when using commits from your branch to add the file type selection:

```
ERROR:root:Error during Graph execution Error on node "Meshing_1":
Log:
ERROR: unrecognised option '--outputDenseFileType'
```

Using the one from my previus comment alone works fine

+😀

natowi commented on 10 Nov 2019 Member

I have changed the code since (check the PRfor details).

At the moment there is the problem left, that

```
class Meshing(desc.CommandLineNode):
    commandLine = 'aliceVision_meshing {allParams}'
```

calls all Params name={...} but my file type selector is not a valid param

```
desc.ChoiceParam(
            name='outputDensePointCloud',
...
```

I think this can be fixed using all supported params, leaving out outputDensePointCloud:

```
class Meshing(desc.CommandLineNode):
    commandLine = 'aliceVision_meshing --input {inputValue} ...
```

I did not test this yet.

+😀

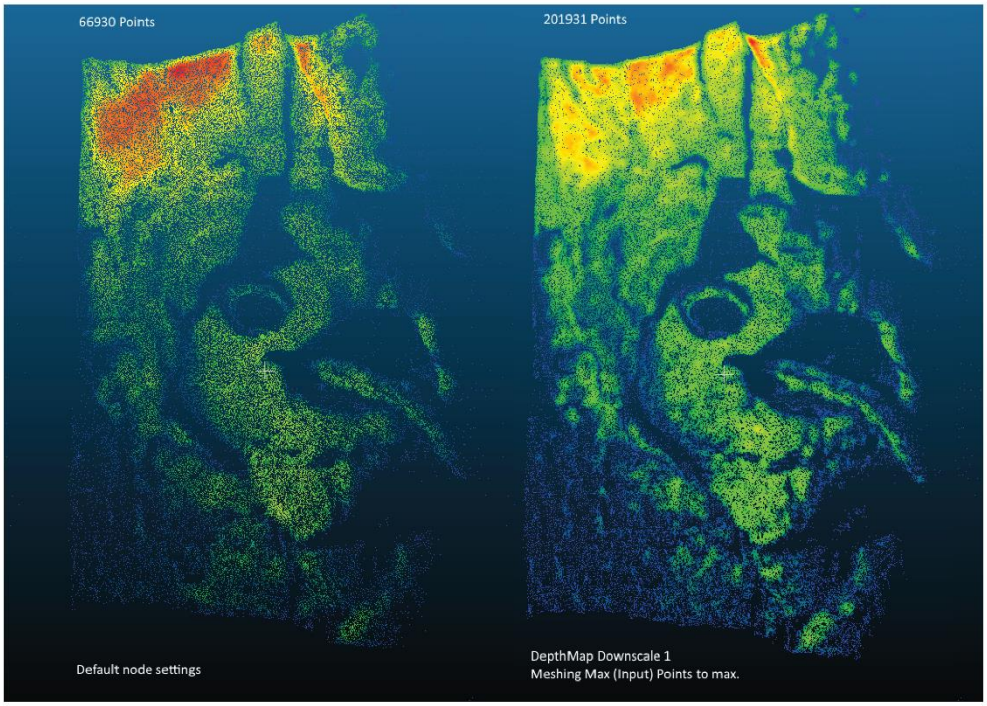
miquelrosell99 commented on 10 Nov 2019 • edited Author

I'll test just to check. What do you mean by the PR? Latest commit is from 2 days ago
Could you send an example for the "commandLine = 'aliceVision_meshing --input {inputValue} ..."?

+😀

 **natowi** commented on 10 Nov 2019 • edited Member

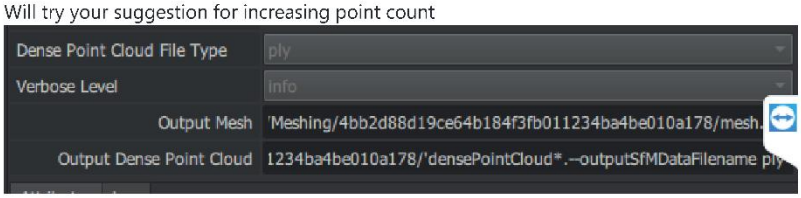
You could try setting downscale to 1 in the DepthMap node and increase the Max Points in Meshing.
(~x3 more points with the monstree demo dataset, 3 images)



2 +😀

miquelrosell99 commented on 10 Nov 2019 • edited Author

@natowi Using your commit for adding a proper menu works great!
This one: 984e76e
Will try your suggestion for increasing point count



+😀

miquelrosell99 commented on 10 Nov 2019 • edited Author

I've taken a different approach to this. I modified the code so it has a .ply file type, but left everything else as is. This way it should work as always, but with .ply file output. After that I use a Publish node, so everything is in the same place if I need to export more stuff
I'll add it to the input section as you say so I can modify the file type, but it still ends up in default folder

```
outputs = [
    desc.File(
        name="outputMesh",
        label="Output Mesh",
        description="Output mesh (OBJ file format).",
        value="{cache}/{nodeType}/{uid0}/mesh.obj",
        uid=[],
    ),
    desc.File(
        name="output",
        label="Output Dense Point Cloud",
        description="Output dense point cloud ('abc', 'ply', 'json', 'xml', 'baf', 'bin').",
        value="{cache}/{nodeType}/{uid0}/densePointCloud.ply",
        uid=[],
    ),
]
```

+😀

natowi commented on 9 Nov 2019 — Member

Yes, this is the easiest solution for your problem

+😀

miquelrosell99 commented on 9 Nov 2019 — Author

Thanks for all the help. With this I can continue working on my project!
Also, any way to get the point could to more than 1 million points? Increasing FeatureExtraction to Ultra generates a denser sparse cloud, but final dense cloud is still 1 million points, same as using FeatureExtraction on Normal

+😀

miquelrosell99 changed the title ~~PrepareDenseScene fails for ConvertSfMFormat in .ply~~ Add option to export dense point cloud to other formats (Workaround here) on 9 Nov 2019

natowi commented on 9 Nov 2019 • edited — Member

miquelrosell99 commented on 9 Nov 2019 • edited ▾ Author

The error is the same

```
C:\Program Files\Meshroom Dev>pip install PySide2
ERROR: Could not find a version that satisfies the requirement PySide2 (from versions: none)
ERROR: No matching distribution found for PySide2

C:\Program Files\Meshroom Dev>
```

PySide install well. But not PySide2

+

natowi commented on 9 Nov 2019 Member

Try running your cli as admin.

+

miquelrosell99 commented on 9 Nov 2019 • edited ▾ Author

Okay, using Python 3.7.5 works. I guess PySide2 is not yet compatible with Python 3.8.0
Everything installed now. I'll try again
Edit: GUI works! Thanks

+

natowi commented on 9 Nov 2019 • edited ▾ Member

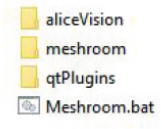
You need to remove the same code from the `outputs = [` section at the end of the file
Basically we move the filepath gui part from output to the input class. (Only the input class allows path modification for now.)

+

miquelrosell99 commented on 9 Nov 2019 • edited ▾ Author

> You need to remove the same code from the `outputs = [` section at the end of the file
> Basically we move the filepath gui part from output to the input class. (Only the input class allows path modification for now.)

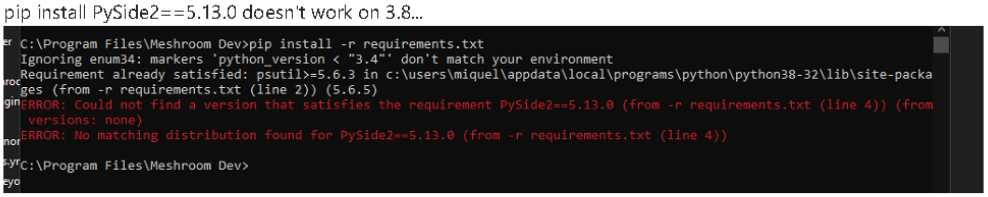
You missed to copy the qtPlugins folder. The best way to put all folders together is to create a new folder and copy paste in the alicevision and qtplugins folders from the binary release and then the meshroom folder from the github repository download.



+😀

miquelrosell99 commented on 9 Nov 2019 • edited ▾ Author

I'm using Python 3.8. Maybe that's the issue. But I can't seem to get pip to install on 3.3
pip install PySide2==5.13.0 doesn't work on 3.8...



+😀

natowi commented on 9 Nov 2019 Member

Do you have pip installed? https://pip.pypa.io/en/stable/installing/

+😀

miquelrosell99 commented on 9 Nov 2019 Author

Yes... Comes preinstalled on 3.8

+😀

natowi commented on 9 Nov 2019 Member

Can you try pip install PySide2 ?

+😀

miquelrosell99 commented on 9 Nov 2019 • edited ▾ Author

@natowi I get this error. I installed the python scripts it told me to install... First command works well btw
C:\Program Files\Meshroom Dev>set PYTHONPATH=%CD% && python meshroom/ui
WARNING:root:== The following plugins could not be loaded ==

- simpleFarmSubmitter: No module named 'simpleFarm'

Attribute Qt::AA_EnableHighDpiScaling must be set before QCoreApplication is created.
WARNING:root:QQmlApplicationEngine failed to load component
WARNING:root:file:///C:/Program Files/Meshroom Dev/meshroom/ui/qml/main.qml:2 plugin cannot be loaded for module "QtQuick.Controls": No se puede cargar la biblioteca C:\Users\Miquel\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-packages\Python37\site-packages\PySide2\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll: No se puede encontrar el m?dulo especificado.

Edit: I also get this error:
ERROR: Could not find a version that satisfies the requirement PySide2 (from versions: none)

+😀

natowi commented on 9 Nov 2019 • edited ▾ Member

Navigate in the extracted meshroom-develop folder,
then install the requirements using

```
cd meshroom-develop
pip install -r requirements.txt
```

or manually install

```
pip install PySide2==5.13.0
```

You might need to install pip first.

@miquelrosell99 you don´t need to build Meshroom, you can use the CLI for now as I described.

1 +

miquelrosell99 commented on 3 Nov 2019 Author

@natowi Yes, that works. Thanks! Eager to see it implemented in GUI too!

+

natowi commented on 8 Nov 2019 • edited Member

Ok here is a work around with gui support

in meshroom/nodes/alicevision/meshing.py add

```
desc.File(
            name="output",
            label="Output Dense Point Cloud",
            description="Output dense point cloud ('abc', 'ply', 'json', 'xml', 'baf', 'bin').",
                    value='',
            uid=[],
        ),
```

as last entry within `inputs = [ ]`

This allows for a custom path to be defined for the dense point cloud.

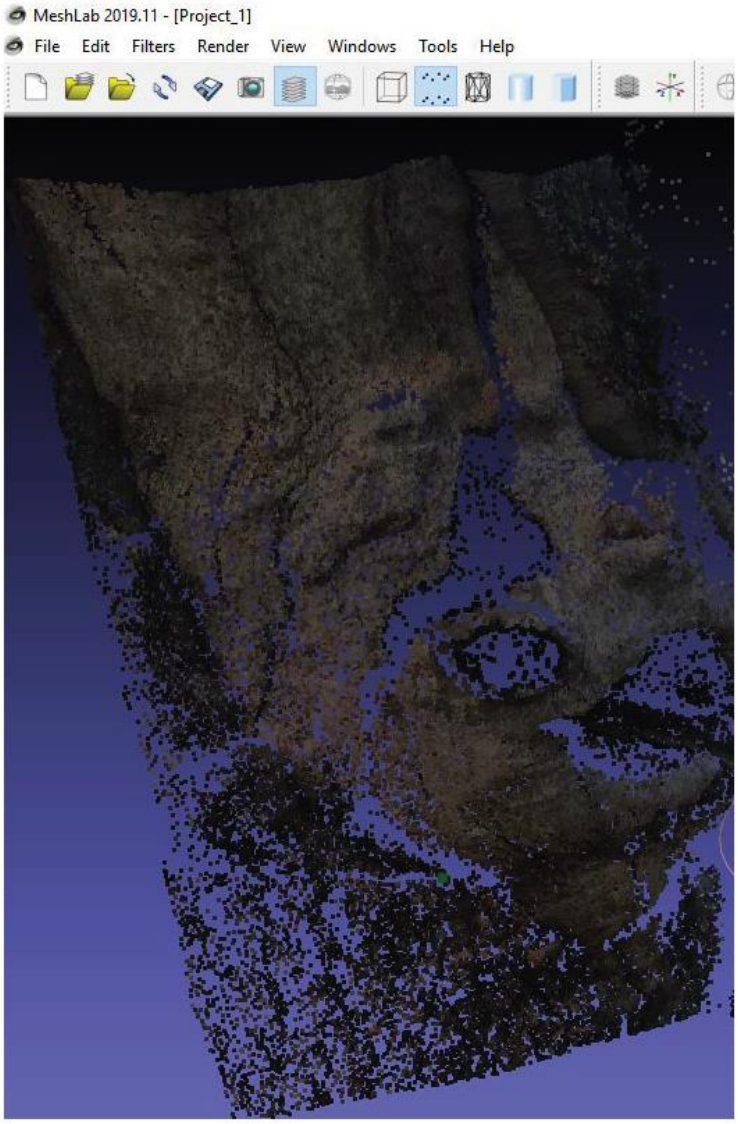
This is how you get the required environment for this to work:

- download the meshroom repository zip from https://github.com/alicevision/meshroom/
- *download the pre-compiled meshroom zip from https://github.com/alicevision/meshroom/releases
- paste the pre-compiled aliceVision and qtPlugins folders in the meshroom folder and use this script to start the meshroom gui (*or use your own compiled versions)
- create a .bat file in the meshroom folder:

```
:: start meshroom bat
set MESHROOM_INSTALL_DIR=%cd%
set PYTHONPATH=%CD% && python meshroom/ui
```

Double click on bat to start meshroom

+



Monstree 3 images dense point cloud demo

I've seen the commits. Can't build meshroom at the moment, so if you do and share via link I'd appreciate it a lot. This way we can test it too. Thanks

natowi commented on 3 Nov 2019 Member

Add filetype selection for dense point cloud ⋯ Verified ed04a64

natowi added a commit that referenced this issue on 2 Nov 2019

Add file type selection for dense point cloud ⋯ Verified 984e76e

natowi mentioned this issue on 2 Nov 2019

[WIP] Add file type selection for dense point cloud #691 Closed

miquelrosell99 commented on 3 Nov 2019 Author

- ConvertSfMFormat at the moment only supports the internal sfm filetype
- Meshing node output filetype for the pointcloud can not be changed at the moment - in the GUI!

Run the Meshing node (only, delete data if necessary) in the GUI with selected settings, then go to the CLI in the background and copy the parameters to a text file (select+enter). Then change abc extension to ply. Open the Meshing folder and delete the files.

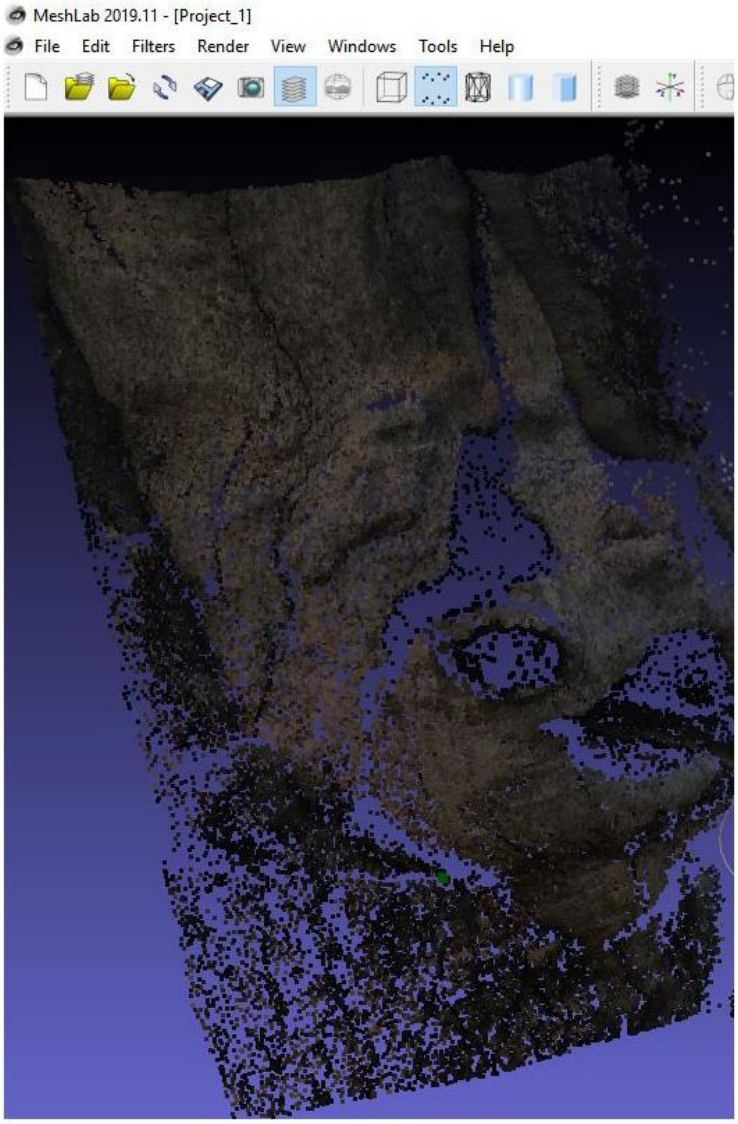
By running the Meshing node now from the CLI we will get our renamed file extension:

Start new CLI, navigate to ..\Meshroom-2019.2.0\aliceVision\bin
then type:

aliceVision_meshing.exe (paste your changed parameters set + enter to run)

Example:

```
C:\Users\user>D:\Meshroom\Meshroom-2019.2.0-win64\Meshroom-
2019.2.0\aliceVision\bin\aliceVision_meshing.exe --input
"C:/Users/user/AppData/Local/Temp/MeshroomCache/StructureFromMotion/5893adbb9647185e705e57e98007c459eb
b29091/sfm.abc" --depthMapsFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMap/79bd8b4b35079d2c5a60be3b5302399952760ed5" --
depthMapsFilterFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMapFilter/47bbddab7c0b3ee1c10675095772b16e22db6a0
1" --estimateSpaceFromSfM True --estimateSpaceMinObservations 3 --estimateSpaceMinObservationAngle 10
--maxInputPoints 50000000 --maxPoints 5000000 --maxPointsPerVoxel 1000000 --minStep 2 --partitioning
singleBlock --repartition multiResolution --angleFactor 15.0 --simFactor 15.0 --pixSizeMarginInitCoef
2.0 --pixSizeMarginFinalCoef 4.0 --voteMarginFactor 4.0 --contributeMarginFactor 2.0 --
simGaussianSizeInit 10.0 --simGaussianSize 10.1 --minAngleThreshold 1.0 --refineFuse True --
addLandmarksToTheDensePointCloud False --colorizeOutput True --saveRawDensePointCloud False --
verboseLevel info --outputMesh
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/mesh.
obj" --output
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/dense
PointCloud.ply"
```



Monstree 3 images dense point cloud demo

Wow thanks so much. Tomorrow I'll look into it and report back

natowi added a commit that referenced this issue on 2 Nov 2019

miquelrosell99 commented on 2 Nov 2019 Author

- ConvertSfMFormat at the moment only supports the internal sfm filetype
- Meshing node output filetype for the pointcloud can not be changed at the moment - in the GUI!

Run the Meshing node (only, delete data if necessary) in the GUI with selected settings, then go to the CLI in the background and copy the parameters to a text file (select+enter). Then change abc extension to ply. Open the Meshing folder and delete the files.

By running the Meshing node now from the CLI we will get our renamed file extension:

Start new CLI, navigate to ..\Meshroom-2019.2.0\aliceVision\bin
then type:

aliceVision_meshing.exe (paste your changed parameters set + enter to run)

Example:

```
C:\Users\user>D:\Meshroom\Meshroom-2019.2.0-win64\Meshroom-
2019.2.0\aliceVision\bin\aliceVision_meshing.exe --input
"C:/Users/user/AppData/Local/Temp/MeshroomCache/StructureFromMotion/5893adbb9647185e705e57e98007c459eb
b29091/sfm.abc" --depthMapsFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMap/79bd8b4b35079d2c5a60be3b5302399952760ed5" --
depthMapsFilterFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMapFilter/47bbddab7c0b3ee1c10675095772b16e22db6a0
1" --estimateSpaceFromSfM True --estimateSpaceMinObservations 3 --estimateSpaceMinObservationAngle 10
--maxInputPoints 50000000 --maxPoints 5000000 --maxPointsPerVoxel 1000000 --minStep 2 --partitioning
singleBlock --repartition multiResolution --angleFactor 15.0 --simFactor 15.0 --pixSizeMarginInitCoef
2.0 --pixSizeMarginFinalCoef 4.0 --voteMarginFactor 4.0 --contributeMarginFactor 2.0 --
simGaussianSizeInit 10.0 --simGaussianSize 10.1 --minAngleThreshold 1.0 --refineFuse True --
addLandmarksToTheDensePointCloud False --colorizeOutput True --saveRawDensePointCloud False --
verboseLevel info --outputMesh
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/mesh.
obj" --output
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/dense
PointCloud.ply"
```

Monstree 3 images dense point cloud demo

natowi added feature request and removed bug labels on 2 Nov 2019

Run the Meshing node (only, delete data if necessary) in the GUI with selected settings, then go to the CLI in the background and copy the parameters to a text file (select+enter). Then change abc extension to ply.
Open the Meshing folder and delete the files.

By running the Meshing node now from the CLI we will get our renamed file extension:

Start new CLI, navigate to ..\Meshroom-2019.2.0\aliceVision\bin
then type:

aliceVision_meshing.exe (paste your changed parameters set + enter to run)

Example:

```
C:\Users\user>D:\Meshroom\Meshroom-2019.2.0-win64\Meshroom-
2019.2.0\aliceVision\bin\aliceVision_meshing.exe --input
"C:/Users/user/AppData/Local/Temp/MeshroomCache/StructureFromMotion/5893adbb9647185e705e57e98007c459ebb29091/sfm.abc" --depthMapsFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMap/79bd8b4b35079d2c5a60be3b5302399952760ed5" --depthMapsFilterFolder
"C:/Users/user/AppData/Local/Temp/MeshroomCache/DepthMapFilter/47bbddab7c0b3ee1c10675095772b16e22db6a01" --estimateSpaceFromSfM True --estimateSpaceMinObservations 3 --estimateSpaceMinObservationAngle 10 --maxInputPoints 50000000 --maxPoints 5000000 --maxPointsPerVoxel 1000000 --minStep 2 --partitioning singleBlock --repartition multiResolution --angleFactor 15.0 --simFactor 15.0 --pixSizeMarginInitCoef 2.0 --pixSizeMarginFinalCoef 4.0 --voteMarginFactor 4.0 --contributeMarginFactor 2.0 --simGaussianSizeInit 10.0 --simGaussianSize 10.1 --minAngleThreshold 1.0 --refineFuse True --addLandmarksToTheDensePointCloud False --colorizeOutput True --saveRawDensePointCloud False --verboseLevel info --outputMesh
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/mesh.obj" --output
"C:/Users/user/AppData/Local/Temp/MeshroomCache/Meshing/709cfa61758574b425bb17aecfc577ca493193c3/densePointCloud.ply"
```

> You can simply change the Inter File Extension from abc to ply in the StructureFromMotion node (Advanced settings enabled) so you don´t need the ConvertSfMFormat node in this case.

But what if I need the dense point cloud? Meshing node only outputs .abc file, and I can't manage to use ConverSfM node there. It only exports the camera dots inside the .ply. No actual object points
Your suggestion was very useful for SfM node. thank you very much. I would really appreciate any help in the issue mentioned above. It's for a final degree project.

+😀

natowi commented on 2 Nov 2019 • edited Member

You can export the Raw Dense Point Cloud by enabling the advanced settings in the Meshing node. You might also want to enable "Colourize Output" to get a coloured point cloud. The output is saved as abc file.

+😀

miquelrosell99 commented on 2 Nov 2019 • edited Author

> You can export the Raw Dense Point Cloud by enabling the advanced settings in the Meshing node. You might also want to enable "Colourize Output" to get a coloured point cloud.

But the raw file is in .abc too. I need .ply
Edit: I need .ply cuz I work with CloudCompare and a Blender plugin that only works with ply. No alembic support in any of the two software

+😀

natowi commented on 2 Nov 2019 Member

Let me check a few things

♡ 1 +😀

natowi commented on 2 Nov 2019 • edited Member

- ConvertSfMFormat at the moment only supports the internal sfm filetype
- Meshing node output filetype for the pointcloud can not be changed at the moment - in the GUI!

Resolved. Will be available in the upcoming release. Added to the documentation.

+ 😀

 **natowi** closed this on 28 Jan

---

Assignees

No one assigned

Labels

type:question

Projects

None yet

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

3 participants